webmasters press

Marco Emrich
Christin Marit

JavaScript

BAND 2

HTML mühelos manipuliert

- Aktuelles DOM für moderne Browser
- Brandaktuell und komplett in ES6-/ECMAScript-2015-Syntax
- Mit vielen Übungen
- Empfehlungen für sauberen Code

Autorisiertes Curriculum für das **Webmasters Europe** Ausbildungs- und Zertifizierungsprogramm

we

Über die Autoren

Marco Emrich

Marco Emrich ist Diplom-Informatiker (FH), leidenschaftlicher Trainer und Verfechter von Software Craftsmanship. Er verfügt über langjährige Erfahrung als Software-Architekt und -Entwickler in ganz unterschiedlichen Branchen. An der Webmasters Akademie in Nürnberg leitet er den Fachbereich Web Engineering. Außerdem hält er regelmäßig Vorträge, leitet Workshops auf bekannten Softwarekonferenzen und schreibt Fachbücher und Artikel für Fachzeitschriften. Wenn er in seiner Freizeit nicht gerade Softwerkskammer-Treffen organisiert, erklärt er wahrscheinlich gerade seinem Sohn, wie man Roboterschildkröten programmiert.

marcoemrich[1]
@marcoemrich[2]

Christin Marit

Christin Marit ist Diplom-Sozialpädagogin, Webentwicklerin, Fotografin, Minimalistin, Weltenbummlerin und professionelle Lebenskünstlerin. Für die Webmasters Akademie in Nürnberg arbeitet Sie als Kursentwicklerin, Autorin und E-Tutorin und findet, dass JavaScriptlernen genauso einfach, schön und spannend sein sollte wie das Leben selbst. Unter simplewinke.de bloggt sie über die Leichtigkeit des Seins.

@christinmarit[3]

---

1. *https://github.com/marcoemrich*
2. *https://twitter.com/marcoemrich*
3. *https://twitter.com/christinmarit*

Marco Emrich und Christin Marit

JavaScript

HTML mühelos manipuliert

Ein Webmasters Press Lernbuch

Version 2.0.4 vom 14.7.2016

Autorisiertes Curriculum für das Webmasters Europe Ausbildungs- und Zertifizierungsprogramm.

www.webmasters-europe.org

Made with Prince Print with CSS

Jetzt Buch registrieren und Begleitmaterial herunterladen!

Registrieren Sie Ihr Buch auf webmasters-press.de und laden Sie sich die Übungsdateien und Lösungen herunter:

www.webmasters-press.de/register
Registrierungscode: 12491a522d4b

© 2016 by Webmasters Press
www.webmasters-press.de

Webmasters Akademie Nürnberg GmbH
Neumeyerstr. 22–26
90411 Nürnberg
Germany
www.webmasters-akademie.de

Printed books made with Prince

Art.-Nr. 12491a522d4b
Version 2.0.4 vom 14.7.2016

Das vorliegende Fachbuch ist urheberrechtlich geschützt. Alle Rechte vorbehalten. Die Verwendung der Texte und Abbildungen, auch auszugsweise, ist ohne schriftliche Genehmigung des Verlags urheberrechtswidrig und daher strafbar. Dies gilt insbesondere für die Vervielfältigung, Übersetzung oder Verwendung in elektronischen Systemen sowie für die Verwendung in Schulungsveranstaltungen. Die Informationen in diesem Fachbuch wurden mit größter Sorgfalt erarbeitet. Trotzdem können Fehler nicht vollständig ausgeschlossen werden. Autoren und Herausgeber übernehmen keine juristische Verantwortung oder irgendeine Haftung für eventuell verbliebene fehlerhafte Angaben und deren Folgen.

Inhaltsverzeichnis

Vorwort 9

1 Was Sie schon immer über das DOM wissen wollten... 11

1.1 DOM-Manipulation im Kontext von Webwendungen 12
1.2 Ein Leben ohne jQuery 13
1.3 JavaScript mit Vanille-Geschmack 14
1.4 Voraussetzungen 15
1.5 Was dieses Buch ist 15
1.6 Was dieses Buch nicht(!) ist 15
1.7 Warum Firefox? 16
1.8 Projekt »NerdWorld« 17

2 Jetzt werden Sie Manipulator 19

2.1 »Hello DOM« mit querySelector und innerHTML 19
2.2 Ein Blick hinter die Kulissen 21
2.3 Vom Suchen und Finden im DOM 25
2.4 ...sie alle zu finden ... — mit querySelectorAll 31
2.4.1 Attribut-Selektoren 35
2.4.2 Combinator Selectors 36
2.4.3 Pseudo Classes 38
2.5 Document vs. Element 39
2.6 Kurz und bündig — $ 40
2.7 Zusammenfassung 42

3 Jetzt geht's ans CSS 43

3.1 Listige Informationen mit classList & DOMTokenList 43
3.2 classList hat Methode(n) 44
3.2.1 Mehr Klasse mit add 44
3.2.2 Was zuviel ist, ist zuviel — mit remove 46
3.2.3 Wechselspiel mit toggle 46
3.2.4 Are you there? — mit contains 47
3.3 Nur highlighten, was man highlighten will — mit filter 47
3.4 Methoden??? 48
3.5 Zusammenfassung 48
3.6 Übungen 50

4 Und nun: ab ins HTML 52

4.1 Rauszögern, ohne abzuwarten: Defer & Async 53
4.2 Strikt und abgeschottet: Blocks & use strict 54
4.3 Nur ein kleines bisschen Refactoring … 55
4.4 Array.from war gestern. Jetzt heißt es: Array-Methoden für alle! 60
4.5 Und jetzt noch eine klitzekleine Verbesserung 61

5 Events feiern, wie sie fallen … 63

5.1 Events im Browser 63
5.2 Zum anständigen Behandeln von Events: Eventhandler 64
5.2.1 Vorbereitende Maßnahmen 64
5.2.2 … und nun alles zusammen 65
5.3 Guard Clauses: Wächter für Ihre Funktionen 67
5.4 Der gekonnte Einstieg — mit init 69
5.5 Schauen Sie hinter die Fassade — mit dem Event-Objekt 70
5.6 preventDefault oder: Wer mag schon immer nur »Standard«? 74
5.7 Kurz und knapp — on 81
5.8 Zusammenfassung 84
5.9 Übungen 87

6 Mehrere JS-Dateien verwenden 89

6.1 Fröhliches Datei-Zerhacken 91
6.2 Der Terror der globalen Verschmutzung 94
6.3 Kein bisschen verstaubte Bibliotheken 96

7 Attribute: Erkennen, Ändern und Manpulieren ausdrücklich erlaubt 98

7.1 IDL-Attribut vs. Content-Attribut 100
7.2 Es lebe der Fortschritt(sbalken)! 100
7.3 Achtung! — boolesche Attribute 101
7.4 Auf die Fragestellung kommt es an — mit dem value-Attribut 103
7.5 Zusammenfassung 105
7.6 Übungen 107

8 Eine Frage des Stils: Das Style-Objekt 110

8.1 Absolut berechnend: getComputedStyle 113
8.2 Noch style-ischer mit Tooltips 119
8.3 Übungen 125

9 HTML unterwandern mit Data-Attributen 129

9.1 Zusammenfassung 130
9.2 Übung 131

10 Von neuen Elementen und angehängten Kindern: createElement & appendChild 133

10.1 HTML generieren lassen mit document.createElement 134
10.2 Endlich angefügt: appendChild 136
10.3 Mal wieder etwas Refactoring 137
10.4 Und noch ein wenig JSON 140
10.5 Vergleich verschiedener DOM-Creation-Methoden 142
10.6 Zusammenfassung 143

11 Ab in die Tonne mit remove 147

11.1 Klassifizierung 151
11.2 Zusammenfassung 152
11.3 Übungen 153

12 Wenn Geschwister eine Reise tun: siblings & insertBefore 154

12.1 Allerliebstes Refactoring 158
12.2 Uups — das geht noch besser! 161
12.3 Zusammenfassung 164
12.3.1 Beispiel zu insertBefore 165
12.3.2 DOM-Traversal 166
12.4 Übungen 168

13 Anhang A: Referenz 172

13.1 Elemente 173
13.1.1 Element vs. Node 173
13.1.2 Der Document-Knoten 173
13.1.3 DOM Selection 174
13.1.4 DOM Manipulation 179
13.1.5 DOM Traversal 180
13.1.6 DOM Creation 181
13.2 Attribute 182
13.3 CSS 185
13.4 Events 186

14 Anhang B: Quellen & Literaturhinweise 189

14.1 APA-Style 189

14.2 Quellen 189

191

Index 208

Vorwort

JavaScript, HTML und CSS

... das sind die drei Säulen, auf denen das moderne Web ruht. Egal, welche Technologien Sie sonst noch verwenden, an dem Erfolgs-Trio führt kein Weg vorbei.

Der erste Band (*JavaScript — Aller Anfang ist leicht*) hat sich mit den Grundlagen der Sprache JavaScript beschäftigt. Im vorliegenden Band zeigen wir Ihnen, wie Sie JavaScript nutzen können um »HTML mühelos zu manipulieren«. Diese Manipulation hilft uns, Webanwendungen zu bauen, die mehr sind als bloße statische Seiten — Webanwendungen, die sich flüssig anfühlen und gut bedienbar sind, statt nach jedem Klick die Seite neu aufbauen zu müssen.

Nach diesem Kurs können Sie dynamische Tooltips erzeugen, Interfaces zum Verwalten von Produkten programmieren oder eigene Bildgalerien entwickeln. Uns würden noch viele andere Anwendungsfälle einfallen — vor allem aber wollen wir Ihnen Werkzeuge an die Hand geben, mit denen Sie die Anwendungsfälle umsetzen können, die uns gerade nicht einfallen.

Die Kombination aus HTML und JavaScript (und natürlich auch CSS) versetzt Sie in die Lage, Ihre Ideen Realität werden zu lassen. Wo fertige Webbaukästen Sie nur einschränken, ist JavaScript ein flexibles und williges Werkzeug Ihrer Kreativität — nur Ihre Phantasie ist die Grenze.

Im vorliegenden Band geht es nicht um Frameworks wie AngularJS oder React. Stattdessen möchten wir Ihnen zeigen, dass eben jener Browser, mit dem Sie Tag für Tag arbeiten, bereits alles mitbringt, was Sie für dynamische Webanwendungen im Frontend benötigen.

Einerseits möchten wir Sie ermutigen, kleinere Probleme direkt mit den Bordmitteln des Browsers zu lösen, statt gleich ein schwergewichtiges und komplexes Framework einzubinden. Andererseits setzen Sie vielleicht später für größere Anwendungen ein solches Framework ein. Dann werden Ihnen die Grundkenntnisse aus diesem Kurs eine wertvolle Hilfe sein. Sie helfen Ihnen, die Arbeitsweise des Frameworks zu verstehen und Probleme auch dann zu lösen, wenn Sie an die Grenzen des Frameworks stoßen.

Wie schon in Band 1 begonnen, setzen wir den aktuellen JavaScript-Standard ein. Sie sollten bereits mit den grundlegenden Spracheigenschaften von ECMAScript2015 vertraut sein. Ansonsten lohnt es sich eventuell, nochmal den Band 1 zur Auffrischung durchzuarbeiten. Wir setzen das dort begonnene Prinzip fort und

verwenden neben den aktuellen Sprachmitteln auch aktuelle Browserimplementierungen. Wir zeigen Ihnen die aktuellen APIs und ein modernes DOM, mit dem das Entwickeln von Webanwendungen wieder Spaß macht. Was sich hinter dem geheimnisvollen Begriff »DOM« verbirgt, erfahren Sie gleich in der ersten Lektion. In diesem Sinne, viel Spaß beim Lesen und Coden…

Christin & Marco

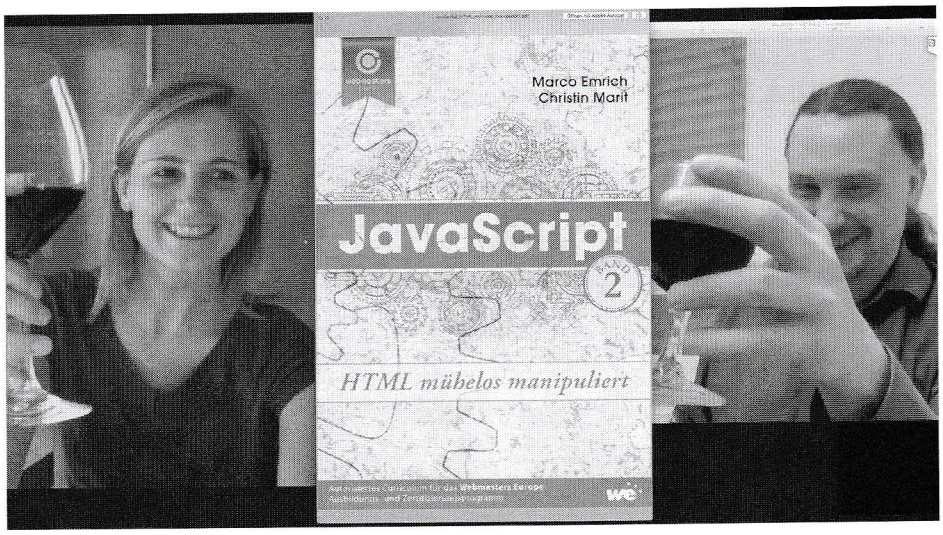


Abb. .1 *Remote Programmieren … Remote Schreiben … und Remote Feiern :)*

Was Sie schon immer über das DOM wissen wollten…

1

> And then there is the DOM — I'm pretty confident saying this: I think, it is the worst API ever invented.
>
> Douglas Crockford *about »Upgrading the Web«*[4]

Vielleicht wollen Sie ja erst einmal wissen, was dieses DOM eigentlich ist, um das es hier geht. Also, das DOM ist kein Gebäude — es ist die Abkürzung für ***Document Object Model*** und der hässliche Star dieses Buches.

Es handelt sich dabei um eine sogenannte ***API*** — ein ***application programming interface*** — eine Schnittstelle zur Programmierung von Anwendungen. Im Klartext heißt das, eine Sammlung von Objekten, Methoden, Funktionen und Attributen. Sie sehen bald, was das genau bedeutet.

Der Browser stellt uns damit eine Möglichkeit zur Verfügung, mit der wir HTML-Seiten dynamisch verändern können. Alle Webanwendungen, die Sie kennen, mit denen Sie ohne ständiges Neuladen arbeiten können, wie Google-Mail, Twitter, Facebook usw., verwenden das DOM, um Ihnen gut bedienbare Oberflächen (darüber lässt sich vermutlich streiten) bereitzustellen. JavaScript (kurz: JS) im Browser ist wertlos ohne die Möglichkeit, HTML zu manipulieren — und dafür benötigen Sie das DOM. Auch moderne Frameworks wie AngularJS oder ReactJS greifen »unter der Haube« auf das DOM zurück.

Warum eigentlich der *hässliche* Star? Douglas Crockford (der Mann hinter JSON und Autor von *JavaScript — the good parts*) hat 2015 in seinem Vortrag *Upgrading the Web* das DOM die schlechteste API aller Zeiten geschimpft.

Schwer zu sagen, ob das stimmt — aber als geplagte Entwickler hat uns das Ringen mit der API schon das eine oder andere graue Haar verursacht (bei Marco ist das ganz deutlich zu sehen). Tatsache ist: Die API ist komplex und nicht einfach zu verwenden.

Die gute Nachricht ist: Das DOM ist vor allem deswegen so komplex, weil es überladen ist — mit vielen, vielen Dingen, die in der Praxis kaum Verwendung finden.

4. *https://www.youtube.com/watch?v=6UTWAEJlhww*

Deswegen greifen wir für Sie die Teile aus der API heraus, die Ihnen den größten Nutzen bringen. Kombiniert mit aktueller ECMAScript6-Syntax und ein paar Kniffen, die wir Ihnen bald zeigen, wird aus der schlechtesten API aller Zeiten … immer noch keine gute API. Aber zumindest gelingt es Ihnen, Ihre Ziele zu erreichen und dabei Code zu produzieren, der auch langfristig lesbar und wartbar bleibt.

1.1 DOM-Manipulation im Kontext von Webwendungen

Wenn Sie vorhaben, größere JS-Anwendungen oder gar ***Single Page Applications*** zu entwickeln, ist reine DOM-Manipulation sicherlich nicht das Mittel der Wahl. Spezialisierte Frameworks wie *AngularJS* oder *ReactJS* sind dafür besser geeignet.

Warum sind wir der Meinung, dass Sie dennoch *zuerst* DOM-Manipulation lernen sollten?

Dafür gibt es eine ganze Reihen von Gründen. Zunächst einmal kann es sein, dass Sie gerade keine Single Page App entwickeln, sondern einfach einer klassische Webanwendung (z. B. einer im *ROCA-Stil*[5]) eine modernere Fassade verpassen wollen. Vielleicht möchten Sie aber auch die Usability verbessern oder einfach ein wenig den »Fancyness«-Faktor erhöhen. Dafür ist ein wenig gezielt und nicht übertrieben eingesetztes ***DOM-Scripting*** oft die richtige Wahl. Sie benötigen keine zusätzlichen Frameworks, müssen keine riesigen Datenmengen nachladen oder komplizierten Setup-Code erstellen. Alles, was Sie benötigen, bringt der Browser mit.

Wo wir gerade über Datenberge sprechen: Vielleicht ist es Ihnen auch wichtig, dass Ihre Website schnell lädt, weil Ihnen ihre mobilen Nutzer am Herzen liegen. Oder Sie wissen, dass Geschwindigkeit ein wichtiger Rankingfaktor für Google ist; ein Gesichtspunkt, den viele SEO-Experten (***Search Engine Optimization***, dt. ***Suchmaschinenoptimierung***) immer noch sträflich vernachlässigen. In diesem Fall ist es besonders wichtig, dass Sie möglichst wenig zusätzlichen Code benötigen.

Sollten Sie später mit einem umfangreicheren Webframework arbeiten, werden Sie feststellen, dass nicht jedes Framework alles kann oder manche »einfachen« Dinge in dem einen oder anderen Framework gar nicht so einfach sind — weil sie gerade nicht ins Frameworkkonzept passen.

Dann ist es an der Zeit, mal wieder selbst Hand anzulegen oder zumindest zu verstehen, was das Framework »unter der Haube« so treibt.

5. *http://roca-style.org/*

Nur wer das DOM beherrscht, beherrscht den Browser.

1.2 Ein Leben ohne jQuery

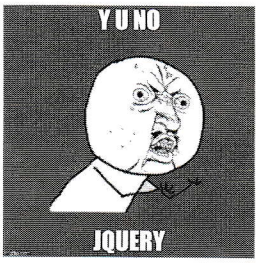


Abb. 1.1 *Y-U-No-jQuery-Meme[6], erstellt mit dem ImgFlip-Memegenerator*

»*Warum zur Hölle verwendet Ihr kein jQuery?*« lautet der aus der Memesprache (sehr verbreitet in der Webenwicklungsszene) übersetzte Ausdruck, den Sie in Abb. 1.1 finden.

Tatsächlich hatten wir dieses Buch ursprünglich als jQuery-Buch begonnen. Es waren sogar schon drei Lektionen fertig. Aber sowohl eigene Projekte als auch Gespräche mit Dutzenden von Entwicklern auf Konferenzen haben bestätigt: Die Zeit ist reif — reif für ein Web ohne jQuery.

Verstehen Sie uns nicht falsch — wir sind keine Gegner von jQuery. jQuery ist eine großartige Bibliothek. Wir haben sie in vielen Projekten gerne eingesetzt. Was jQuery vor allem geleistet hat, war, eine stabile Plattform zu bieten, die die Unterschiede zwischen den Browsern ausgleicht. jQuery hat jede noch so verrückte Browserbesonderheit vor uns verborgen und uns so viel Ärger und Tausende von Entwicklungsstunden erspart.

jQuery brachte auch zu Zeiten von EcmaScript 3.1 bereits High-Order-Funktionen wie `map` und `each` mit. Vermutlich hat jQuery sogar den EcmaScript-Standard dazu inspiriert, ähnliche Funktionalität direkt in die Sprache zu integrieren.

Ich kann gar nicht beschreiben, wie verrückt die Web-Welt war, bevor Bibliotheken wie jQuery und Prototype sich um Browserunterschiede kümmerten. Ich hab' oft nächtelang Dokumentationen gewälzt und JS-Code debuggt, um herauszufinden, warum gerade diese eine Sache, die in allen anderen Browsern tadellos funktioniert, in IE5/IE6 mal wieder anders ist.

6. *https://imgflip.com/i/r7vc2*

Jedenfalls entwickelt sich das Web weiter. Unter der Voraussetzung, dass der *Internet Explorer 9* der älteste Browser ist, den Sie noch unterstützen müssen, sind die Unterschiede zwischen den Browsern nicht mehr so groß. Es kommt natürlich auf Ihre Zielgruppe und Ihren Anwendungsfall an, aber die Notwendigkeit jQuery (oder vergleichbare Bibliotheken) einzusetzen, ist nicht mehr so groß wie noch vor wenigen Jahren.

Mittlerweile hat sich eine ganze Community um das Thema *#nojquery*[7] (Twitter Hashtag) gebildet. Hier ein paar Links zum Weiterschmökern:

- *youmightnotneedjquery.com*
- *blog.garstasio.com/you-dont-need-jquery*
- *www.sitepoint.com/do-you-really-need-jquery*
- *lea.verou.me/2015/04/jquery-considered-harmful*
- *tutorialzine.com/2014/06/10-tips-for-writing-javascript-without-jquery*
- *programmers.stackexchange.com/questions/166273/advantages-of-using-pure-javascript-over-jquery*

Unter *youmightnotneedjqueryplugins.com* finden Sie sogar eine Liste von jQuery-freien Widgets und Plugins. Direkte Vergleiche zwischen jQuery und Vanilla-Code (erklären wir gleich) finden Sie in Ray Nicholus' Buch (2015) *Beyond jQuery*.

1.3 JavaScript mit Vanille-Geschmack

Generell ist die Idee dieses Buches, mit ***vanilla JS*** zu arbeiten, so lange es sinnvoll ist. Der Begriff *vanilla JS* bedeutet: einfaches JavaScript ohne Schnickschnack oder Beiwerk. Als Belohnung winken kleine Dateigrößen und hohe Performance.

Ein weiterer Begriff mit der gleichen Bedeutung wie vanilla JS wäre übrigens *naked JS* (Nur hat uns das Lektorat leider alle darauf aufbauenden Scherze gestrichen).

Unter *vanilla-js.com* finden Sie eine Satireseite von Eric Wastl, die *vanilla JS* wie eine Bibliothek oder ein Framework anpreist, das Sie sich herunterladen können — eine leere Datei :)

> *The Vanilla JS team takes pride in the fact that it is the most lightweight framework available anywhere; using our production-quality deployment strategy, your users' browsers will have Vanilla JS loaded into memory before it even requests your site.*
>
> vanilla-js.com

7. *https://twitter.com/search?q=%23nojquery*

Ignoriert man mal den humoristischen Aspekt, so bietet die Seite vor allem interessante Vergleiche zwischen vanilla JS und beliebten Bibliotheken — Vergleiche, die zeigen, dass »reines« JS nun wirklich keine schlechte Wahl ist.

1.4 Voraussetzungen

Wir gehen davon aus, dass Sie entweder Band 1: »*JavaScript — Aller Anfang ist leicht*« durchgearbeitet haben oder äquivalente JavaScript-Grundkenntnisse mitbringen. Sollten Ihnen Konzepte wie Variablen, Kontrollstrukturen (z. B. `if`), Rekursion oder High-Order-Funktionen (`map`, `reduce`, ...) nicht vertraut sein, empfehlen wir Ihnen, sich erst mal mit den Grundlagen zu beschäftigen. Der Band 1 dieser Reihe: *JavaScript — Aller Anfang ist leicht* (Emrich, Marit 2015) eignet sich dafür übrigens hervorragend :)

> Dieses Buch verwendet konsequent die Sprachversion ***ES6*** bzw. ***EcmaScript 2015*** von JavaScript.

Sie sollten also auch mit Basis-ES6-Features wie Stringtemplates, `let`, `const` oder dem *fat arrow* `=>` vertraut sein.

1.5 Was dieses Buch ist

Dieses Buch konzentriert sich auf JavaScript im Browser. Inbesondere behandeln wir das Browser-DOM und zeigen Beispiele, wie sie in echten Projekten tatsächlich vorkommen. D. h. Sie lernen die Browser-Funktionen anhand von Code kennen, wie wir ihn (nahezu) auch in Wirklichkeit schreiben würden. Deswegen erwartet Sie kein Feuerwerk an Features, die wir im Akkord abhandeln. Stattdessen zeigen wir Ihnen, wie Sie die neuen Funktionen und Konzepte sinnvoll in realen Umgebungen einsetzen. Dazu gehören auch oft viele Verbesserungsrunden, um schließlich Code zu präsentieren, auf den Sie stolz sein können.

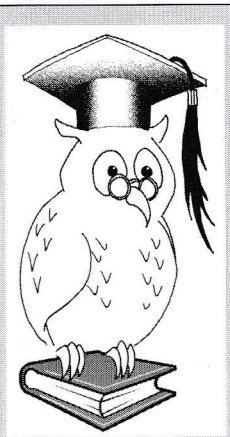
1.6 Was dieses Buch nicht(!) ist

Wenn Sie später Software mit JS entwickeln, werden Sie oft Details nachschlagen müssen. Dafür ist dieses Buch aber nur sehr bedingt geeignet. Wir versuchen eher, Ihnen einen Leitfaden an die Hand zu geben, der Ihnen einen schnellen Einstieg in das Browser-DOM ermöglicht. Dabei ist die Reihenfolge der Lektionen auf den Lernfortschritt ausgerichtet, und nicht auf schnelles Auffinden von Themen optimiert. Wir erheben auch keinen Anspruch auf Vollständigkeit, sondern beschrän-

ken uns bewusst auf die Dinge, die für Sie als Einsteiger zunächst am wichtigsten sind — das »Big Picture« sozusagen.

Wenn Sie später bei Ihrer täglichen Entwicklungsarbeit alle Details benötigen, empfehlen wir Ihnen folgende Websites:

- *www.mozdev.org* — Mozilla Developer Network (MDN)
  Das MDN ist viel mehr als eine Referenz für den Firefox-Browser. Sie finden hier sowohl ausführliche Details zum JS-Sprachkern als auch zur Browser-Funktionalität. Selbst Tabellen, welcher Browser welches Feature ab welcher Version implementiert, sind vorhanden — häufig mit Polyfills (Code, der das Feature nachrüstet) für den Fall der Fälle.
- *www.webplatform.org*
  Ein gemeinsames Dokumentationsprojekt vom W3C, Microsoft, Adobe und anderen. Sehr gute und ausführliche Dokumentation zu HTML5 und CSS. Die JS-Dokumentation ist noch in Arbeit, aber schon vielversprechend.
- *caniuse.com*
  Kann ich ein bestimmtes Feature benutzen? Diese Site kennt die Antwort. Ausführliche Übersichten zu neuen Features in HTML, CSS und JS mit Angaben zu den Browserversionen und Hinweisen zu Polyfills.



Falls Sie nach JS-Inhalten googeln möchten, ist es ratsam, ein »mdn« zu ergänzen, damit Sie gleich auf der passenden Seite des Mozilla Developer Networks landen — sonst besteht die Gefahr, dass Sie Seiten in der Trefferliste finden, deren Erklärungen veraltet oder schlichtweg falsch sind.

1.7 Warum Firefox?

Wir setzen Mozilla-Firefox (Version 46+, englische Fassung) als Beispielbrowser für diesen Kurs ein. Firefox hat den Vorteil, dass er bereits alle ES6-Features unterstützt, die wir verwenden.

Im Grunde spricht nichts dagegen, wenn Sie lieber einen anderen Browser verwenden möchten. Sie sollten nur sicherstellen, dass der Browser bereits über die

eingesetzten ES6-Features verfügt. Eine Übersicht über den Stand der ES6-Unterstützung finden Sie in der *ES6 compatibility table*[8].

1.8 Projekt »NerdWorld«

Im ersten Band haben wir das Projekt *NerdWorld* mit seinem Auftraggeber *Björn* vorgestellt. NerdWorld ist ein fiktives Projekt, anhand dessen Sie viel über JS lernen können — vor allem auch, wo und wie Sie JS sinnvoll einsetzen.

Auch dieses Mal hat Björn wieder jede Menge für Sie zu tun und stellt Sie vor interessante Herausfordungen. Herausfordungen, die Sie Dank JavaScript natürlich **mühelos** bewältigen!

Wenn Sie das Projekt bereits aus dem Buch *JavaScript - Aller Anfang ist leicht* kennen, können Sie den folgenden Abschnitt überspringen. Ansonsten wiederholen wir hier noch mal die Projektbeschreibung.

Ein Kundengespräch

Stellen Sie sich vor, Ihr erster Kunde ist *Björn* — ein Ladenbesitzer, der seine Produkte nun auch im Internet anbieten möchte. Der Laden heißt »*NerdWorld*« (das Beispiel ist fiktiv — Sie können sich das Googeln sparen).

NerdWorld ist ein Online-Shop, bei dem Nerds und andere Technik-Verrückte Produkte des »täglichen Bedarfs« kaufen: Nerfguns, Ufos mit USB-Anschluss, Klingonenwaffen fürs Büro, koffeinhaltige Getränke, stark koffeinhaltige Getränke, noch stärker koffeinhaltige Getränke usw. Eben alles, was der moderne Entwickler zum Überleben im Büro-Dschungel so braucht.

Es gibt zwar schon eine statische HTML-Site, nur können Sie dort bisher keine Produkte kaufen. Die neue Site soll primär aus einem Shop bestehen. Um das Ganze etwas interessanter zu gestalten, sind außerdem weitere Features geplant oder sollen weiterentwickelt werden, wie z. B.

- ein Chat-Client, über den sich die Kunden mit den Verkäufern/Kundenberatern austauschen
- eine Newsletter-Funktion, mit der der Shop die Kunden über neue Produkte informiert
- ein Newsboard, auf dem die Kunden die neuesten Neuigkeiten erfahren.

8. *http://kangax.github.io/compat-table/es6/*

Damit dieser Traum auch Realität wird, hat sich der Geschäftsführer an einen Spezialisten gewandt: an Sie! Sie treffen sich also mit ihm, und er schildert Ihnen die Situation:

Seit Jahren verkaufen wir cooles Zeug für Nerds — oder solche, die es werden wollen. Unsere Produktpalette reicht von Star Trek-Fanartikeln über Programmierer-Utensilien bis hin zu smartphonegesteuerten Mini-Drohnen und Scherzartikeln wie dem Elektroschock-Kugelschreiber.

Im Laufe dieses Kurses werden wir immer wieder auf *Projekt NerdWorld* zu sprechen kommen und Ihnen zeigen, wie Sie die Anforderungen mit JS und dem DOM bewältigen.

Jetzt werden Sie Manipulator 2

2.1 »Hello DOM« mit querySelector und innerHTML

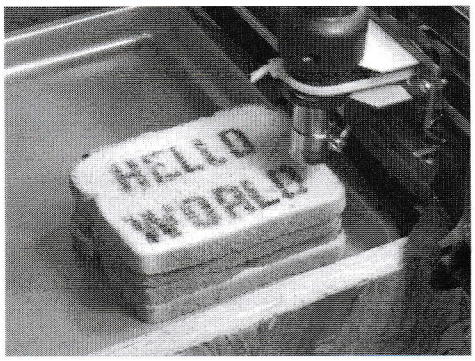


Abb. 2.1 *Foto: oskay*[9]
Lizenz: CC BY 2.0[10]

Jedes ernstzunehmende Buch über Programmierung beginnt mit einem *Hello World*-Beispiel. Natürlich wollen wir mit dieser Tradition nicht brechen. In Band 1 haben Sie *Hello World* auf der Konsole ausgegeben. In diesem Buch geht es nun um die HTML-Struktur. Also »sagen« Sie erst einmal *Hello World* auf einer HTML-Seite. Sie benötigen dafür:

1. eine einfache HTML-Seite (Codebeispiel 2.1)
2. Die JavaScript-Konsole des Browsers

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="utf-8" />
6    <title>Hello World</title>
7  </head>
8
9  <body>
10
11   <h1>Hello <em>World</em></h1>
```

9. *https://www.flickr.com/photos/oskay/472097903/sizes/o/in/photostream*
10. *http://creativecommons.org/licenses/by/2.0*

```
12
13 </body>
14
15 </html>
```

Codebeispiel 2.1 *Einfaches HTML-Document*

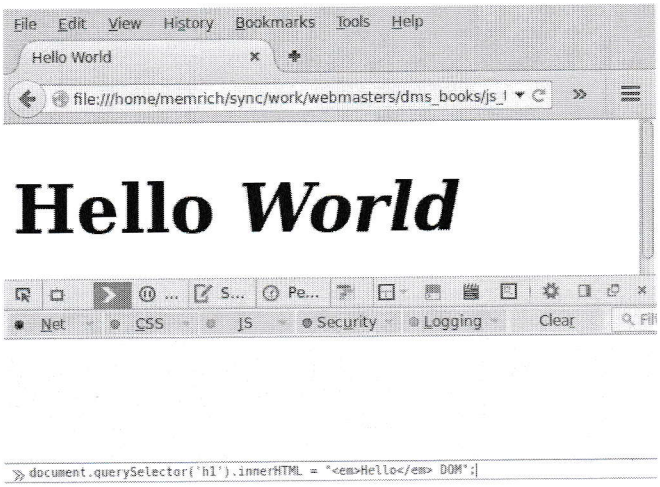


Abb. 2.2 *Hello World in Firefox*

Schritt für Schritt:

1. Öffnen Sie das HTML-Dokument aus Codebeispiel 2.1 in Firefox. Die Datei finden Sie im Begleitmaterial zum Buch unter *02/examples/simple.html*

2. Öffnen Sie die Web-Konsole (**ctrl-shift-K**) und geben Sie folgenden Code in die JavaScript-Befehlszeile ein (Abb. 2.2):

```
document.querySelector('h1').innerHTML = "<em>Hello</em>
DOM";
```

Sobald Sie die Zeile mit **Enter** bestätigen, ändert sich die Überschrift von *Hello World* nach *Hello DOM* — mit entsprechend geändertem Markup (siehe Abb. 2.3).

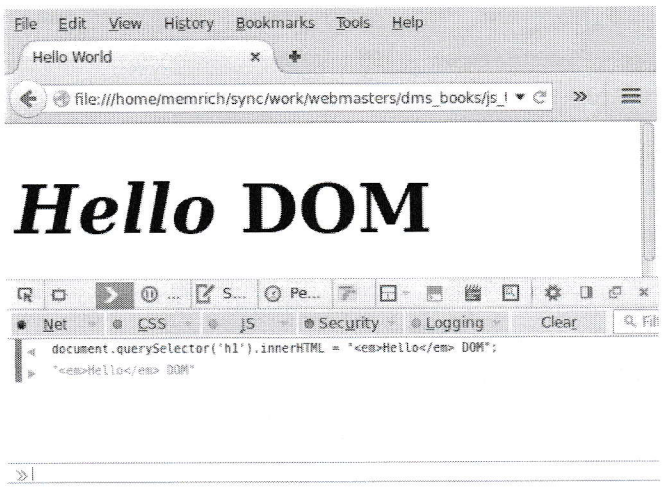


Abb. 2.3 *Hello DOM in Firefox*

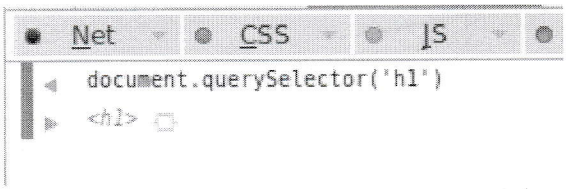
2.2 Ein Blick hinter die Kulissen

Wie funktioniert das?

Sehen Sie sich das Ganze nochmal in Zeitlupe an. Na gut, wir haben leider keine richtige Zeitlupe, aber wir können zurückspulen (mit einem Browser-Reload) und uns die beiden Teile der Anweisung als Einzelschritte ansehen.

3. Laden Sie also die HTML-Seite neu (**ctrl-R** bzw. **cmd-R**), um wieder zur ursprünglichen *Hello World*-Seite zurückzukehren.

4. Geben Sie nur `document.querySelector('h1')` (ohne Zuweisung danach) ein.



Mit der Funktion `document.querySelector` können Sie HTML-Elemente innerhalb des HTML-Dokumentes auswählen. Der Parameter ist ein beliebiger CSS-Selektor. So liefert Ihnen `document.querySelector('h1')` das erste `h1`-Element als JS-Objekt zurück.

Bisher haben wir den Begriff ***Objekt*** noch nicht genauer definiert. Im Moment ist aber nur wichtig, dass Objekte Werte sind, wie Zahlen oder Strings. Objekte können auch komplexere Dinge repräsentieren, wie hier z. B. das `h1`-Element.

Im Übrigen können Sie das `h1`-Element in der Konsole untersuchen. Beim Mouse-Over wird es im HTML-Dokument hervorgehoben. Klicken Sie das Symbol daneben an, zeigt der Firefox-Inspector, wo es sich im HTML-Baum befindet (siehe Abb. 2.4).

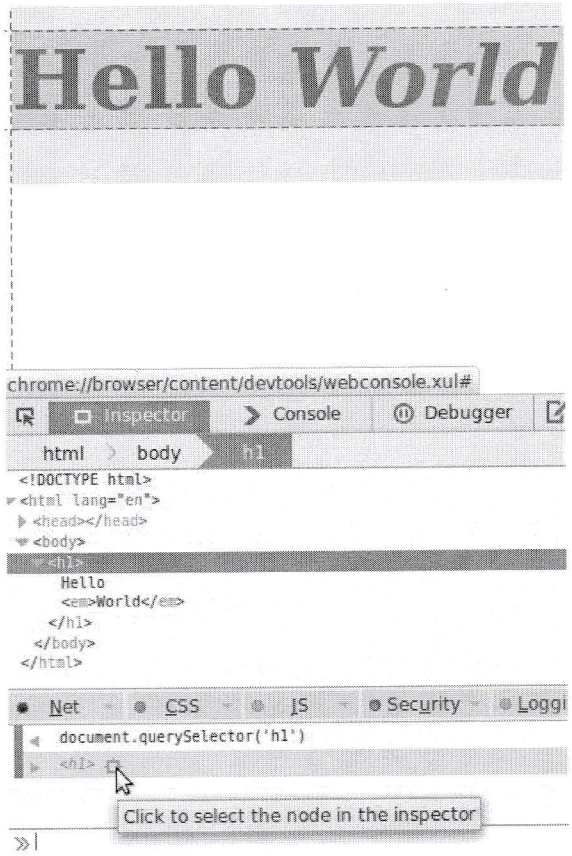


Abb. 2.4 *h1 im Firefox-Inspector*

Der Rückgabewert der Funktion `document.querySelector` ist also ein JS-Objekt, das ein Element im HTML-Baum repräsentiert. Genauer gesagt ist es ein sogenanntes Element-Objekt. Objekte in JS haben ***Eigenschaften*** (engl. ***properties***). Eine wichtige Eigenschaft, speziell von Element-Objekten, ist *innerHTML*. Sie ermitteln ihren Wert mit Hilfe der Punkt-Notation, z. B. `document.querySelector('h1').innerHTML`

5. Geben Sie jetzt in die Konsole ein:

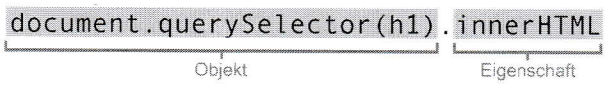
```
document.querySelector('h1').innerHTML
```

Sie erhalten als Ausgabe:

```
Hello <em>World</em>
```

`document.querySelector('h1')` liefert das Element-Objekt zurück, dessen Eigenschaft *innerHTML* Sie abfragen. Der Wert von `innerHTML` ist ein String mit dem *inneren HTML* des jeweiligen Elementes — also dem HTML-Quellcode, der sich innerhalb der Tags befindet.

Die Punktnotation funktioniert auch mit anderen Objekten & Eigenschaften. Die Schreibweise hat dabei immer die Form: `<Objekt>.<Eigenschaft>`



Sie können Eigenschaften ähnlich wie Variablen verwenden. Genauso wie bei einer Variable können Sie auch Eigenschaften neue Werte zuweisen. Sobald Sie der Eigenschaft `innerHTML` einen neuen Wert zuweisen, ändert sich die HTML-Seite.

6. Geben Sie ein:

```
document.querySelector('h1').innerHTML = "Hello again!";
```

Damit Sie uns auch glauben, dass Objekte tatsächlich einfach nur Werte sind, sind wir Ihnen noch einen Beweis schuldig. Hier ist er: Sie können ein Objekt, wie jeden anderen Wert, einfach an einen Variablennamen binden.

7. Geben Sie ein:

```
let myH1 = document.querySelector('h1');
myH1.innerHTML = "Hello once more!";
```

Außerdem liefert Ihnen nun die Eingabe von `myH1` in der Konsole wieder das HTML-Element-Objekt zurück.

8. Probieren Sie es aus! Geben Sie einfach nur ein:

```
myH1
```

Übung 1: Almost Famous Quotes

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="utf-8" />
6
7   <title>Famous Quotes</title>
8   <meta name="description" content="JavaScript. HTML mühelos manipuliert" />
9
10   <link rel="stylesheet" href="../../css/styles.css" type="text/css" media="screen" />
11   <link rel="shortcut icon" href="../../img/favicon.ico" type="image/x-icon" />
12 </head>
13
14 <body>
15
16   <header></header>
17
18   <main>
19     <h1>Famous Quotes</h1>
20
21     <blockquote></blockquote>
22   </main>
23
24 </body>
25
26 </html>
```

Codebeispiel 2.2 *additional_files/02/examples/quotes.html*

1. Ändern Sie mit Hilfe von JS die Überschrift (`h1`) der HTML-Seite aus Codebeispiel 2.2 in *Almost Famous Quotes*.

2. Die Seite (Codebeispiel 2.2) enthält ein leeres `blockquote`-Element. Öffnen Sie die HTML-Seite in Firefox. Geben Sie in der Konsole eine Zeile JS-Code ein, die das Element mit folgendem Inhalt befüllt:

```
<p>I have always wished for my computer to be as easy to
use as my telephone; my wish has come true because I can
no longer figure out how to use my telephone.</p>
<footer>— <cite>Bjarne Stroustrup</cite></footer>
```

2.3 Vom Suchen und Finden im DOM

Sie haben gesehen, dass `document.querySelector` HTML-Elemente als JS-Objekte extrahieren kann. Tatsächlich ist das ganze HTML-Dokument nichts anderes als eine Sammlung von HTML-Elementen, die in einer Baumstruktur angeordnet sind. Äquivalent dazu sind die passenden JS-Objekte ebenfalls in einem Baum angeordnet — dem sogenannten ***Document Object Model***, kurz ***DOM***. Für die einfache HTML-Seite aus Codebeispiel 2.3 sieht das DOM beispielsweise wie in Abb. 2.6 aus.

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="utf-8" />
6    <title>DOM-Example</title>
7    <link rel="stylesheet" href="css/styles.css" type="text/css"
media="screen" />
8  </head>
9
10 <body>
11
12   <header>
13     <nav>
14       <ul>
15         <li></li>
16         <li></li>
17         <li></li>
18       </ul>
19     </nav>
20   </header>
21
22   <main>
23     <article>
24       <h1></h1>
25       <p></p>
```

```
26        <img alt="" src="#" />
27      </article>
28    </main>
29
30    <footer>
31      <nav></nav>
32    </footer>
33
34  </body>
35
36  </html>
```

Codebeispiel 2.3 *additional_files/02/examples/dom_example.html*

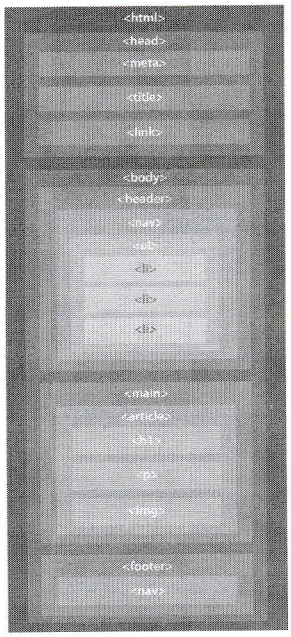


Abb. 2.5 *Visualisierung der verschachtelten HTML-Struktur*

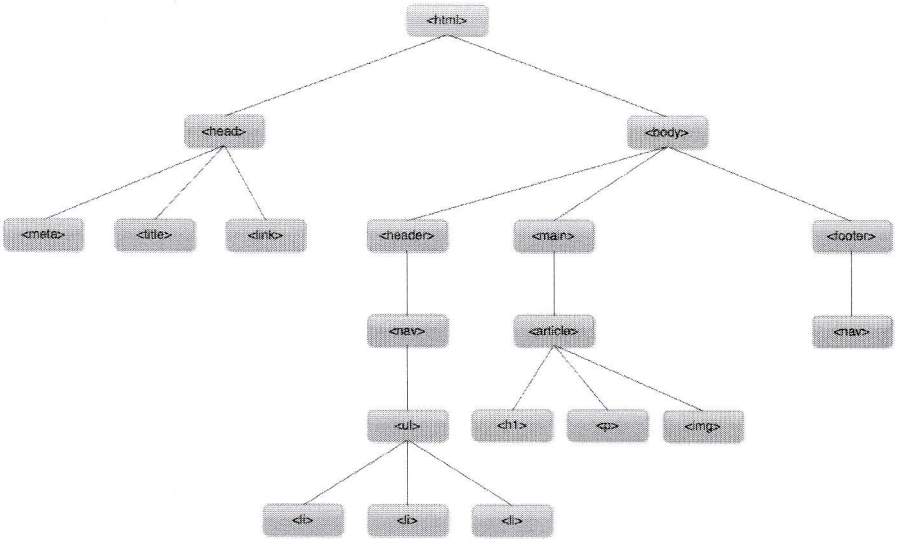


Abb. 2.6 *Baumstruktur der HTML-Seite*

Die Funktion `document.querySelector` selektiert immer das erste Element, das dem übergebenen CSS-Selektor entspricht, und gibt es zurück. Der Selektor kann dabei ein beliebiger valider CSS-Selektor sein. Tabelle 2.1 zeigt die wichtigsten.

Name	Beispiel	Beschreibung
Universal selector	`*`	Selektiert ein beliebiges Element. Diesen Selektor sollten Sie nicht ohne weitere Einschränkungen verwenden, da das Selektieren **aller** Elemente einer Website sehr viel Rechenzeit in Anspruch nehmen kann.
Type selector	`h1`	Selektiert alle Elemente eines bestimmten Typs.
ID selector	`#some-id`	Selektiert genau **ein** Element mit der gegebenen `id`.

Tabelle 2.1 *Basis CSS3-Selektoren*

Name	Beispiel	Beschreibung
Class selector	`.some-class`	Selektiert alle Elemente, denen die angegebene Klasse zugewiesen wurde. Beachten Sie, dass ein Element mehrere Klassen haben kann.
-	`h1, p, .go`	Das Komma ist selbst kein Selektor, sondern erlaubt es, mehrere Selektoren zu kombinieren. Im Beispiel würden alle `h1`-Elemente, alle `p`-Elemente und alle Elemente mit der Klasse `go` selektiert.

Tabelle 2.1 *Basis CSS3-Selektoren*

Übung 2: 01001000010011110101010 0 — Teil 1

Abb. 2.7 *Hot Binary Heat Changing Mugthinkgeek, © 2016 ThinkGeek, Inc. All Rights Reserved.*[11]

Jetzt sind Sie dran. Schreiben Sie Ihre eigenen Abfragen. Benutzen Sie das folgende HTML-Dokument als Übungsplattform.

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="utf-8" />
6
7   <title>01001000010011110101010 0</title>
8   <meta name="description" content="JavaScript, HTML mühelos
manipuliert" />
9
```

11. *http://www.thinkgeek.com/product/2024/?srp=6*

```
10   <link rel="stylesheet" href="../../css/styles.css"
type="text/css" media="screen" />
11   <link rel="shortcut icon" href="../../img/favicon.ico"
type="image/x-icon" />
12 </head>
13
14 <body>
15
16   <header></header>
17
18   <main>
19     <h1 class="article">Hot Binary Heat Changing Mug <span
class="keyword">"01001000010011110101010 0"</span></h1>
20
21     <img alt="Hot Binary Heat Changing Mug" src="../../img/
thinkgeek_2024_hot_binary_heat_change_mug.gif"
class="float_left" id="product_img" />
22
23     <h2>Description</h2>
24
25     <p>Numbers make up <span
class="special">everything</span> in our digital world. They
flow around us, invisible like the Force or the Matrix,
controlling all our many computer-y devices. Two numbers, in
particular: <span class="special">0 and 1</span>. <span class="b
i">Off and On</span>. Well, we can tell you this: when there's
no coffee in our cup, we're completely OFF our game. But when
our mug is full of hot coffee, we're totally ON. And now, with
the <span class="keyword">Hot Binary Heat Changing Mug</span>,
there's a mug that tells us which state our mug is in. In
binary!</p>
26
27     <p>See, the <span class="keyword">Hot Binary Heat
Changing Mug</span> looks like just a dark mug with binary
numbers all over it. That's its OFF or cold state. Add hot
coffee (or any liquid) and a series of digits will turn white.
Read them continuously from left to right, and you'll read:
<span class="keyword">01001000010011110101010 0</span>. That's
<span class="special">in binary</span>. Of course, your <span
class="keyword">Hot Binary Heat Changing Mug</span> could just
be paying you a compliment. Cheeky, mug.</p>
28
29     <p><img alt="01001000010011110101010 0" src="../../img/
thinkgeek_2024_hot_binary_heat_change_mug_grid_embed.jpg" /></p>
30
31     <h2>Product Specifications</h2>
32
33     <ul id="product_specification">
34       <li class="keyword">Hot Binary Heat Changing Mug</li>
35       <li>As you add hot liquids, the binary for "HOT"
appears (read from left to right in one line, not two:
01001000010011110101010 0)</li>
36       <li>A <span class="keyword">ThinkGeek</span> creation
and exclusive!</li>
37       <li>Care Instructions: <span class="i b">Hand wash
```

```
only. Not microwave or dishwasher safe.</span></li>
38        <li>Materials: Ceramic</li>
39        <li>Dimensions: approx. 3.15" diameter x 3.75" tall</li>
40      </ul>
41
42      <h3>You wanna buy it?</h3>
43
44      <p class="buy_info_text">Wenn Sie unsere sehr geniale
Tasse kaufen wollen, folgen Sie einfach den folgenden
Schritten:</p>
45
46      <ol class="model" data-model="LDV73C-X3">
47        <li>Select how many items do you want.</li>
48        <li>Press "buy".</li>
49      </ol>
50
51      <form id="buy_form">
52        <select>
53          <option>1 item</option>
54          <option>2 items</option>
55          <option>3 items</option>
56          <option>4 items</option>
57        </select>
58        <input type="button" value="buy" />
59      </form>
60    </main>
61
62    <footer>(C) by ThinkGeek &ndash; Produkttext mit
freundlicher Genehmigung von ThinkGeek Inc.</footer>
63
64 </body>
65
66 </html>
```

Codebeispiel 2.4 *additional_files/02/exercises/0100100001001111010101 00.html*

Tippen Sie eine Abfrage in die Konsole, die:

1. das `h1`-Element findet.
2. das Element mit der `id` *buy_form* findet.
3. das Element mit der `id` *product_img* findet.

```
only. Not microwave or dishwasher safe.</span></li>
38        <li>Materials: Ceramic</li>
39        <li>Dimensions: approx. 3.15" diameter x 3.75" tall</li>
40      </ul>
41
42      <h3>You wanna buy it?</h3>
43
44      <p class="buy_info_text">Wenn Sie unsere sehr geniale
Tasse kaufen wollen, folgen Sie einfach den folgenden
Schritten:</p>
45
46      <ol class="model" data-model="LDV73C-X3">
47        <li>Select how many items do you want.</li>
48        <li>Press "buy".</li>
49      </ol>
50
51      <form id="buy_form">
52        <select>
53          <option>1 item</option>
54          <option>2 items</option>
55          <option>3 items</option>
56          <option>4 items</option>
57        </select>
58        <input type="button" value="buy" />
59      </form>
60    </main>
61
62    <footer>(C) by ThinkGeek &ndash; Produkttext mit
freundlicher Genehmigung von ThinkGeek Inc.</footer>
63
64 </body>
65
66 </html>
```

Codebeispiel 2.4 *additional_files/02/exercises/01001000010011110101010 0.html*

Tippen Sie eine Abfrage in die Konsole, die:

1. das `h1`-Element findet.
2. das Element mit der `id` *buy_form* findet.
3. das Element mit der `id` *product_img* findet.

2.4 ...sie alle zu finden ... — mit querySelectorAll

Auf unserer News-Seite haben alle Artikel eine eigene Überschrift. Ergänzen Sie bitte ein `Aktuell:` vor jeder dieser Überschriften.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <title>Headlines</title>
  <link rel="stylesheet" href="../../css/styles.css" type="text/
css" media="screen" />
</head>

<body>
  <h2>Lorem ipsum dolor sit amet.</h2>
  <h2 class="news">Quod eligendi saepe voluptates eveniet.</h2>
  <h2 class="news">Architecto reiciendis magnam modi
inventore.</h2>
  <h2 class="news">Modi ipsum, velit rem ipsam.</h2>
  <h2 class="news">Provident, officia quisquam aliquam
deserunt!</h2>
</body>

</html>
```

Codebeispiel 2.5 *additional_files/02/examples/news.html*

Es gibt oft Situationen, in denen Sie mehrere Elemente gleichzeitig auslesen oder verändern müssen. Für diese Fälle gibt es die Funktion `querySelectorAll`. Im Gegensatz zu `querySelector` liefert sie, statt des ersten Treffers, **alle** zum Selektor passenden Elemente zurück.

Öffnen Sie die Seite *news.html* (Codebeispiel 2.5) und geben Sie folgenden Code in die Konsole ein.

```
document.querySelectorAll('h2')
```

Sie erhalten:

```
NodeList [ <h2>, <h2.news>, <h2.news>, <h2.news>, <h2.news> ]
```

Eine ***NodeList*** ist ein Objekt, das mehrere Elemente (Knoten/Nodes) in einer Liste zusammenfasst. Der Begriff *Knoten* ist dabei im Sinne des HTML-Baums gemeint. Ein Knoten hat immer nur ein Eltern-Element, kann aber beliebig viele (oder auch gar keine) Kind-Elemente haben.

Die *NodeList* verhält sich ähnlich wie ein Array. Sie können mit dem Index-Operator darauf zugreifen. So gibt Ihnen z. B.

```
document.querySelectorAll('h2')[0].innerHTML
```

den Text der ersten `h2`-Überschrift zurück, also im Beispiel den Text *Lorem ipsum dolor sit amet.*

Wenn Sie wissen möchten, wie viele Knoten der Selektor gefunden hat, oder ob überhaupt ein Element zum Selektor passt, können Sie, wie bei einem Array, die Eigenschaft `length` abfragen:

```
document.querySelectorAll('h2').length // => 5
```

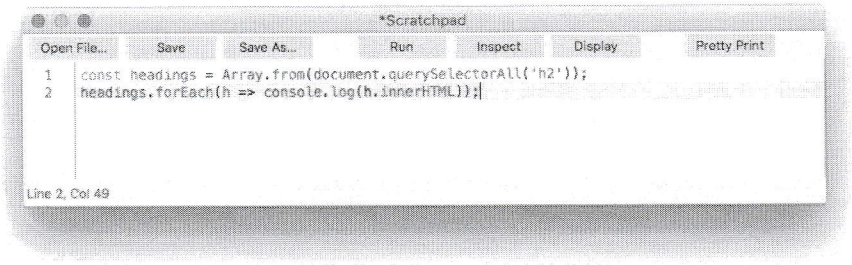
```
document.querySelectorAll('img').length === 0 // => true, means no
images found
```

Um nun Björns Anforderung zu erfüllen, genügt es natürlich nicht, die Überschriften nur zu selektieren. Sie müssen die Überschriften ändern, oder besser gesagt, den Text innerhalb der Überschriften-Tags.

Dazu müssen Sie die Überschriften aus der *NodeList* mit einem `forEach` durchlaufen. `forEach` ist aber nur für Arrays definiert — nicht für NodeLists. Zum Glück lässt sich eine NodeList mittels `Array.from` in ein Array konvertieren.

```
1 const headings = Array.from(document.querySelectorAll('h2'));
2 headings.forEach(h => console.log(h.innerHTML));
```

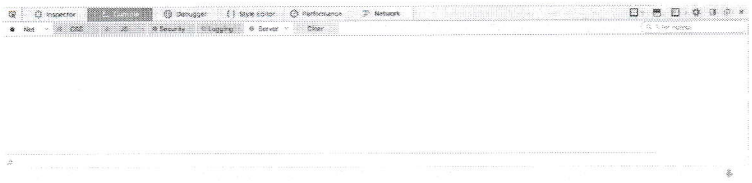
Es handelt sich hier übrigens um einen mehrzeiligen Code, den Sie sicherlich speichern möchten: Daher ist es besser, wenn Sie ab jetzt Scratchpad (**shift-f4**) statt der Konsole verwenden.



Statt die Überschriften nur auszugeben, sollten Sie diese nun ändern.

```
1 const headings = Array.from(document.querySelectorAll('h2'));
2 headings.forEach(h => h.innerHTML = 'Aktuell: ' + h.innerHTML);
```

```
Aktuell: Lorem ipsum dolor sit amet.
Aktuell: Quod eligendi saepe voluptates eveniet.
Aktuell: Architecto reiciendis magnam modi inventore.
Aktuell: Modi ipsum, velit rem ipsam.
Aktuell: Provident, officia quisquam aliquam deserunt!
```



Übung 3: 01001000010011110101010 0 — Teil 2

Basteln Sie noch etwas an der Artikel-Seite weiter. Schreiben Sie Ihre eigenen Abfragen. Benutzen Sie wieder das HTML-Dokument aus Codebeispiel 2.4 als Übungsplattform.

Tippen Sie eine Abfrage in die Konsole, die:

1. alle `li`-Elemente findet.
2. alle `h2`-Elemente findet.

3. alle Elemente mit der Klasse *special* findet.
4. alle `li`-Elemente mit der Klasse *keyword* findet.
5. alle `span`-Elemente mit der Klasse *special* findet.
6. alle Elemente findet, die die Klassen *i* UND *b* haben.

Übung 4: Makler und Anwälte...

Abb. 2.8 *additional_files/02/exercises/makler_und_anwaelte.html*

Stellen Sie sich vor, Sie haben die fertige Website einer Immobilien-Firma in der »Schublade«. Nun haben Sie einen neuen Kunden — eine Anwaltskanzlei — und finden, dass sich das Layout der Website der Makler auch perfekt für die Anwälte eignen würde. Natürlich können Sie dem neuen Kunden den »neuen« Entwurf nicht mit den Texten der Makler vorstellen. Damit Sie nicht mühevoll mit copy & paste hantieren müssen, schreiben Sie Code, der die vorhandenen Texte gegen **Blind-Texte** austauscht. Sie können dafür einen der berüchtigten Lorem Ipsum-Generatoren (oder natürlich das geniale *Bacon Ipsum*[12] bzw. *Veggie Ipsum*[13] — je nach persönlichem Geschmack) verwenden.

1. Schreiben Sie eine Funktion, die den Inhalt aller `h1`-Tags austauscht.

12. *https://baconipsum.com*
13. *http://veggieipsum.com*

2. Schreiben Sie eine weitere Funktion, um den Inhalt der `p`-Tags auszutauschen.
3. Auch die Navigationspunkte sollen ausgetauscht werden.

2.4.1 Attribut-Selektoren

Die letzte Übung war eine nette Fingerübung, aber es gibt noch viel mehr CSS-Selektoren — diese sind natürlich, wie alle CSS-Selektoren, auch in JS verwendbar.

Die nächsten CSS-Selektoren, die Sie sich anschauen sollten, sind die sogenannten ***attribute selectors***. Mit ihrer Hilfe können Sie auf Elemente mit bestimmten Attributen und Werten zugreifen. Tabelle 2.2 zeigt eine Übersicht.

Selektor	Beispiel zu `<img src="funny_cat.png" ... />`	Beschreibung
[attr]	`[src]`	Selektiert Elemente mit dem angegebenen Attribut. Der Attributwert spielt dabei keine Rolle (***attribute presence***).
[attr=val]	`[src='funny_cat.png']`	Selektiert Elemente, die das angegebene Attribut (*src*) mit genau dem angegebenen Wert (*funny_cat.png*) enthalten (***exact attribute match***).
[attr=val]*	`[src*=cat]`	Selektiert Elemente, bei denen das angegebene Attribut (*src*) den Wert (*cat*) als Teilstring enthält (***substring attribute match***).
[attr^=val]	`[src^=funny]`	Selektiert Elemente, bei denen das angegebene Attribut (*src*) mit dem Wert (*funny*) beginnt.
[attr$=val]	`[src$=png]`	Selektiert Elemente, bei denen das angegebene Attribut (*src*) auf den angegebenen Wert (*png*) endet.

Tabelle 2.2 *CSS attribute selectors. All examples select *

2.4.2 Combinator Selectors

Als Nächstes werfen wir einen Blick auf die ***combinator selectors***. Sie erlauben Ihnen, Elemente anhand ihrer Beziehung zu anderen Elementen zu finden. Das einfache Leerzeichen, das Sie vermutlich schon kennen, ist ein solcher combinator. Sie können damit die Nachfahren eines angegebenen Elementes selektieren. In Tabelle 2.3 finden Sie weitere combinator selectors.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <title>Selector Testing</title>
</head>

<body>
  <ul id="ul1">
    <li id="li11"><span>11</span></li>
    <li id="li12">12</li>
    <li id="li13">13</li>
    <li id="li14">14</li>
  </ul>
  <ul id="ul2">
    <li id="li21"><span>21</span></li>
    <li id="li22"><strong><span>22</span></strong></li>
    <li id="li23">23</li>
  </ul>
</body>

</html>
```

Codebeispiel 2.6 *additional_files/02/examples/selector_testing.html*

Name	Beispiel	Beschreibung
Descendant combinator (Leerzeichen)	`li span`	Selektiert Elemente (hier `span`), die die Nachfahren eines anderen angegebenen Elements (hier `li`) sind — völlig unabhängig davon, wie viele Ebenen dazwischen liegen. **Ergebnis:** `[<span>11</span>, <span>21</span>, <span>22</span>]`
Adjacent sibling combinator (+)	`li#li11 + li`	Mit diesem Selektor können Sie das Element auswählen, das im HTML-Quelltext **direkt nach** dem angegeben Element (hier `li#li11`) auf der gleichen Ebene (Geschwister-Element) folgt. Falls das im Dokument nachfolgende Element nicht dem hinter dem +-Symbol angegebenen entspricht, wird nichts selektiert. **Ergebnis:** `[<li id="li12">12</li>]`
General sibling combinator (~)	`li#li12 ~ li`	Mit diesem Selektor können Sie alle Elemente auswählen, die im Dokument nach dem angegebenen Element (hier `li#li12`) auf der gleichen Ebene (Geschwister-Element) folgen. Es wird **nicht nur das direkt folgende** Element gewählt, wie beim *Next sibling combinator*. **Ergebnis:** `[<li id="li13">13</li>, <li id="li14">14</li>]`
Child combinator (>)	`li > span`	Eine Variante der Nachfahren-Selektoren sind die Kind-Selektoren, mit denen Sie Elemente auswählen können, die **direkte** »Kinder« von übergeordneten Eltern-Elementen sind. Im Beispiel werden nur `span`-Elemente selektiert, die sich direkt innerhalb eines `li`-Elements befinden. `<span>22</span>` befindet sich in einem `strong`-Element und wird deswegen nicht selektiert. **Ergebnis:** `[<span>11</span>, <span>21</span>]`

Tabelle 2.3 *Combinator Selectors — Ergebnisse basieren auf Codebeispiel 2.6*

Wie Sie sehen, sind CSS-Selektoren praktisch und einfach zu verwenden. Vermutlich kennen Sie die meisten davon schon längst. Es sollte also nicht schwer sein, sich ein paar weitere zu merken.

Übung 5: 01001000010011110101010 0 — Teil 3

Wenden Sie sich nochmal der 01001000010011110101010 0-Tasse zu. Benutzen Sie geeignete Selektoren, um

1. das erste `li` zu finden, das sich innerhalb der `ul` mit der `id` *product_specification* befindet.
2. das erste `span`-Element zu finden, das sich innerhalb des ersten `h1`-Elements mit der CSS-Klasse *article* befindet.
3. alle Elemente mit der Klasse *keyword* innerhalb von `p`-Elementen zu finden.
4. alle `li`-Elemente innerhalb von `ul`-Elementen zu finden.

2.4.3 Pseudo Classes

Die sogenannten *pseudo classes* erlauben eine weitere Spezialisierung der Selektoren.

Name	Beispiel	Beschreibung
:first-child	ul li:first-child	Mit dieser Pseudo-Klasse sprechen Sie das erste Kind-Element eines übergeordneten Containers (hier: `ul`) an, falls es sich um ein Element vom angegeben Tag-Typ (hier: `li`) handelt. Sollte das erste Kindelement nicht den richtigen Tag-Typ haben, wird nichts selektiert.
:nth-child(x)	ul li:nth-child(5)	Die Pseudo-Klasse spricht das x-te Kind-Element an — im Beispiel das fünfte Element innerhalb von `ul`, falls es vom Tag-Typ `li` ist.
:last-child	ul li:last-child	Äquivalent zu `:first-child` selektieren Sie hier das letzte Kind-Element, sofern es vom richtigen Tag-Typ ist.
:only-child	#article p:only-child	Diese Klasse selektiert ein Element nur, wenn es sich um das einzige Kind eines angegebenen Eltern-Elementes handelt.

Tabelle 2.4 *Pseudo Classes*

Name	Beispiel	Beschreibung
:empty	div:empty	Mit dieser Pseudo-Klasse sprechen Sie nur Elemente an, die ohne Kind-Elemente sind. Inhalt in Form von Text wird dabei ebenfalls als Kind-Element betrachtet.
:not(selector)	:not(span)	Negation. Wählt Elemente aus, wenn sie nicht dem in Klammern angegebenen Selektor entsprechen. Im Beispiel würde alles außer `span`-Elementen ausgewählt.

Tabelle 2.4 *Pseudo Classes*

Die vorgestellten Selektoren sind natürlich längst nicht alle, aber damit kommen Sie in der Praxis schon recht weit. Fall es nötig sein sollte, finden Sie weitere Selektoren in der offiziellen *W3C-specification*[14] oder in einer Online-Dokumentation wie *MDN*[15] oder *Webplatform*[16].

2.5 Document vs. Element

Die Methoden `querySelector` und `querySelectorAll` lassen sich auf `document` **global** und auf einzelnen Elementen **lokal** verwenden. Global heißt, der Selektor durchsucht das gesamte Dokument. Lokal bedeutet, dass nur die Kinder des entsprechenden Element-Knotens zu Suche herangezogen werden.

Beispiel

HTML-Dokument:

```
...
<body>
  <ul>
    <li>1</li>
    <li>2</li>
  </ul>
  <ul id="second_list">
```

14. *http://www.w3.org/TR/css3-selectors/#attribute-selectors*
15. *https://developer.mozilla.org/en-US/docs/Web/CSS/Reference*
16. *http://docs.webplatform.org/wiki/css/selectors*

```
    <li>3</li>
    <li>4</li>
  </ul>
</body>
```

JavaScript in der Konsole:

```
document.querySelectorAll('li')
  // => <li>1</li><li>2</li><li>3</li><li>4</li>
$('#second_list').querySelectorAll('li')
  // => <li>3</li><li>4</li>
```

Codebeispiel 2.7 *// => zeigt den Rückgabewert*

2.6 Kurz und bündig — $

Viele Bibliotheken und Frameworks aus dem JS-Umfeld, wie z. B. *jQuery*[17] und *Prototype*[18], bringen eine Funktion namens `$` mit. Diese Funktion entspricht mehr oder weniger `document.querySelector` bzw. `document.querySelectorAll` — mit kleinen Unterschieden je nach Implementierung. Da Sie `document.querySelectorAll` in der Praxis sehr häufig verwenden, wäre die kürzere Schreibweise `$` extrem nützlich. Ein einzelnes Zeichen ist nicht nur schneller zu tippen, sondern verbessert auch die Lesbarkeit, vor allem bei langen Code-Zeilen.

Um von der kürzeren Schreibweise zu profitieren, müssen Sie sich aber nicht erst eine komplette Bibliothek oder ein großes Framework in Ihr Projekt reinholen. Sie erreichen den gleichen Effekt mit einer simplen Umbenennung. Binden Sie die Funktionen `document.querySelector` und `document.querySelectorAll` einfach an die beiden Bezeichner `$` und `$$`:

```
1 const $  = document.querySelector.bind(document);
2 const $$ = document.querySelectorAll.bind(document);
```

Das `.bind(document)` ist leider notwendig. Das liegt an einer kleinen Unschönheit der Sprache, auf die wir an dieser Stelle nicht näher eingehen möchten.

Fügen Sie diese beiden Zeilen in Ihren Code ein, stehen Ihnen `$` und `$$` als Kurzschreibweisen zur Verfügung.

17. *https://jquery.com*
18. *http://prototypejs.org/*

Übrigens bringen die Konsolen von Firefox und Chrome die Funktionen `$` und `$$` bereits mit — leider wirklich nur für die Konsole. Aber auch das ist oft bereits sehr praktisch.

Übung 6: 01001000010011110101010100 — Teil 4

Wenden Sie sich nochmal der 01001000010011110101010100-Tasse zu. Benutzen Sie geeignete Selektoren, um

1. alle Bilder zu finden, deren Dateiname auf `jpg` endet.
2. alle `input`-Elemente vom Typ `button` zu finden, die sich innerhalb von Formularen befinden.
3. alle Elemente mit der Klasse *model* zu finden, die ein Attribut *data-model* haben, das den Wert *V7* beinhaltet.
4. alle Bilder zu finden, die **nicht** die Klasse `float_left` enthalten.
5. alle **zweiten** Listenpunkte zu finden.
6. alle Listen (`ul`) zu finden, die unmittelbar nach einer zweiten Überschrift (`h2`) folgen.

2.7 Zusammenfassung

Methode	Beschreibung	Interface/API
querySelector	gibt das **erste** Element zurück, das zum angegebenen Selektor passt. Durchsucht nur Elemente innerhalb des HTML-Teilbaums, auf dessen Wurzelelement `querySelector` aufgerufen wurde.	Element
querySelectorAll	gibt **alle** Elemente zurück, die zum angegeben Selektor passen. Durchsucht nur Elemente innerhalb des HTML-Teilbaums, auf dessen Wurzelelement `querySelectorAll` aufgerufen wurde.	Element
document.querySelector *(aka $)*	gibt das **erste** Element zurück, das zum angegeben Selektor passt. Durchsucht das gesamte HTML-Dokument.	Document
document.querySelectorAll *(aka $$)*	gibt **alle** Elemente zurück, die zum angegeben Selektor passen. Durchsucht das gesamte HTML-Dokument.	Document
document.body	gibt das `body`-Element zurück.	Document

Tabelle 2.5 *Diese Tabelle zeigt die wichtigsten Möglichkeiten zur* ***Selektion*** *von DOM-Elementen. Weitere finden Sie im MDN bei den Beschreibungen der entsprechenden APIs: Element*[19] *& Document*[20].

19. *https://developer.mozilla.org/en-US/docs/Web/API/Element*
20. *https://developer.mozilla.org/en-US/docs/Web/API/Document*

Jetzt geht's ans CSS

3

Unser Chat-Room erfreut sich gerade großer Beliebtheit. Wenn allerdings gerade mal wieder sehr viele Kunden online sind, kann die Liste recht unübersichtlich werden. Deswegen wünschen sich einige Anwender die Möglichkeit, direkt nach ihren Freunden zu suchen. Dazu möchten sie den gesuchten Namen in einem Suchfeld eingeben. Dann sollen alle Fundstellen in der Liste hervorgehoben werden.

Wow — das ist keine einfache Aufgabe, was Björn da von Ihnen verlangt. Aber jedes Problem wird überschaubar, wenn Sie es in Einzelteile zerlegen.

3.1 Listige Informationen mit classList & DOMTokenList

```
1 <ul id="chat_members">
2   <li class="admin">Heribert</li>
3   <li>Friedlinde</li>
4   <li>Tusnelda</li>
5   <li>Berthold</li>
6   <li>Oswine</li>
7   <li>Ladislaus</li>
8 </ul>
```

Codebeispiel 3.1 *Ausschnitt aus additional_files/03/examples/chatteilnehmer.html*

Ermitteln Sie zunächst die `li`-Elemente mit den Teilnehmernamen.

```
$$('#chat_members li') // => NodeList [ <li.admin>, <li>, <li>,
<li>, <li>, <li> ]
```

Wie Sie sehen, ist der erste Chat-Teilnehmer, *Heribert*, zusätzlich mit der Klasse `admin` gekennzeichnet.

Um nun einfach mal Heribert als `highlighted` zu kennzeichnen, greifen Sie ihn gezielt aus der *NodeList* heraus:

```
$$('#chat_members li')[0] // => <li class="admin">
```

Jedes `NodeObject` hat neben der Eigenschaft `innerHTML` auch eine Eigenschaft namens `classList`, die die CSS-Klassen des Objekts repräsentiert. Fragen Sie diese Eigenschaft für das erste Element (d.h. Index `0`) ab:

```
$$('#chat_members li')[0].classList // => DOMTokenList [ "admin" ]
```

Sie erhalten eine sogenannte *DOMTokenList* zurück, die nur die Klasse *admin* enthält. Das ist wenig verwunderlich: Heribert ist einer der Administratoren des Chats — dementsprechend hat sein `li`-Element die Klasse *admin*.

Eine *DOMTokenList* verhält sich dabei ähnlich wie ein Array. Sie können z.B. die Eigenschaft `length` verwenden, um die Anzahl der CSS-Klassen zu ermitteln:

```
$$('#chat_members li')[0].classList.length // => 1
```

Friedlinde hat überhaupt keine Klasse:

```
$$('#chat_members li')[1].classList.length // => 0
```

Mit dem Indexoperator `[]` können Sie sich die erste Klasse als String herausziehen:

```
$$('#chat_members li')[1].classList[0] // => 'admin'
```

3.2 classList hat Methode(n)

Einen Unterschied zwischen *DOMTokenList* und Array stellen jedoch die dazugehörigen Funktionen dar. Es gibt kein `push`, `pop`, `sort` usw. Das liegt vor allem daran, dass die Reihenfolge der CSS-Klassen keine Rolle spielt. Statt dessen gibt es die Funktionen `add`, `remove`, `toggle` und `contains`.

3.2.1 Mehr Klasse mit add

Fügen Sie mal spaßeshalber Heribert die Klasse `hightlighted` hinzu:

```
1 $$('#chat_members li')[0].classList.add('highlighted');
```

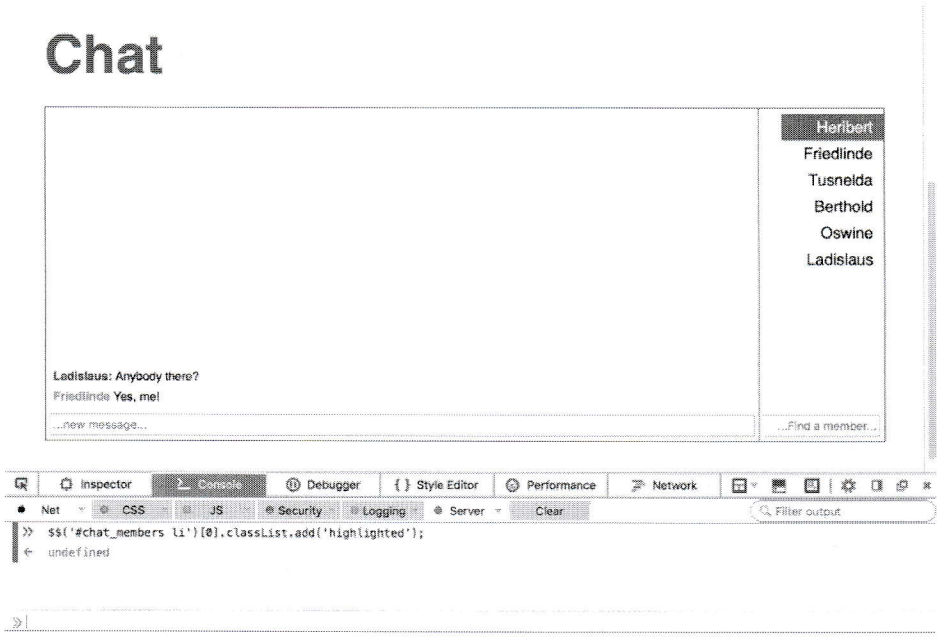


Abb. 3.1 *Chat: »Heribert« hervorgehoben*

Wie Sie in Abb. 3.1 sehen können, ist Heribert nun hervorgehoben. Außerdem beinhaltet die `classList` zwei Elemente:

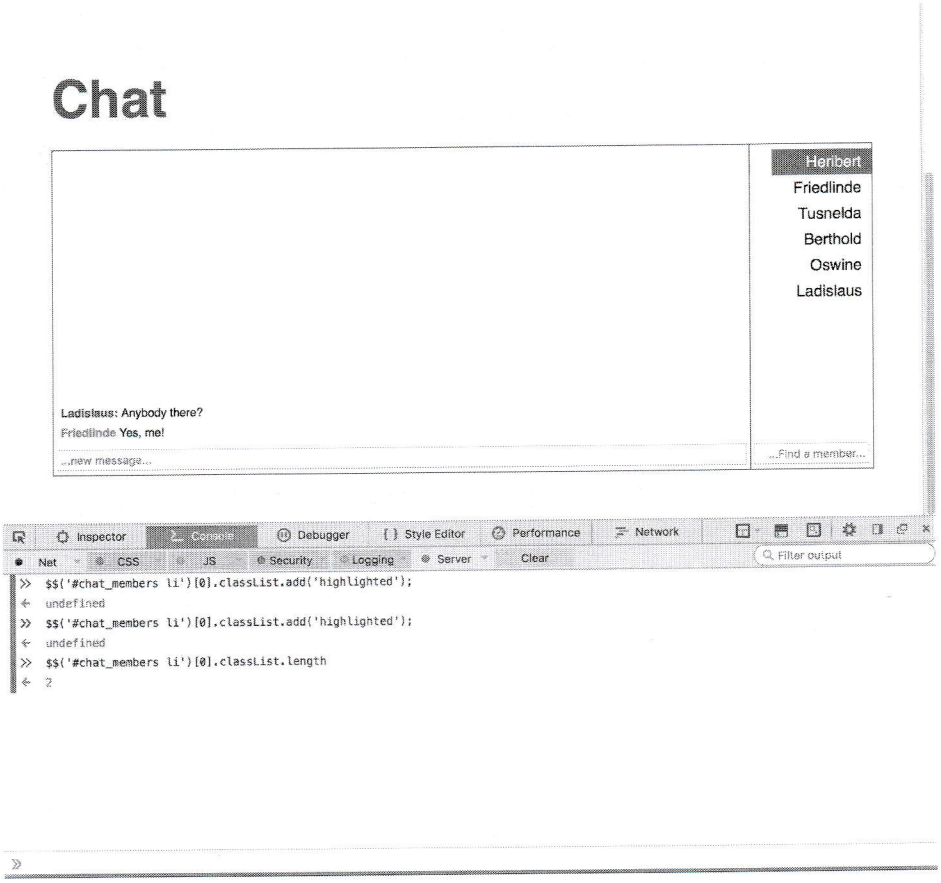
```
1 $$('#chat_members li')[0].classList.length // => 2
```

Wenn Sie Heribert nun im Firefox-Inspector betrachten, sehen Sie, dass das `li`-Element über beide Klassen (*admin* und *highlighted*) verfügt.

Einen weiteren Unterschied erkennen Sie, wenn Sie eine bereits vorhandene Klasse ein weiteres Mal hinzufügen:

```
1 $$('#chat_members li')[0].classList.add('highlighted');
2 $$('#chat_members li')[0].classList.length // => 2
```

Die *DOMTokenlist* hat immer noch nur zwei Elemente. In einer *DOMTokenList* kann im Gegensatz zu einem Array der gleiche Wert nicht mehrfach vorkommen.



Einen weiteren Unterschied erkennen Sie, wenn Sie eine bereits vorhandene Klasse ein weiteres Mal hinzufügen:

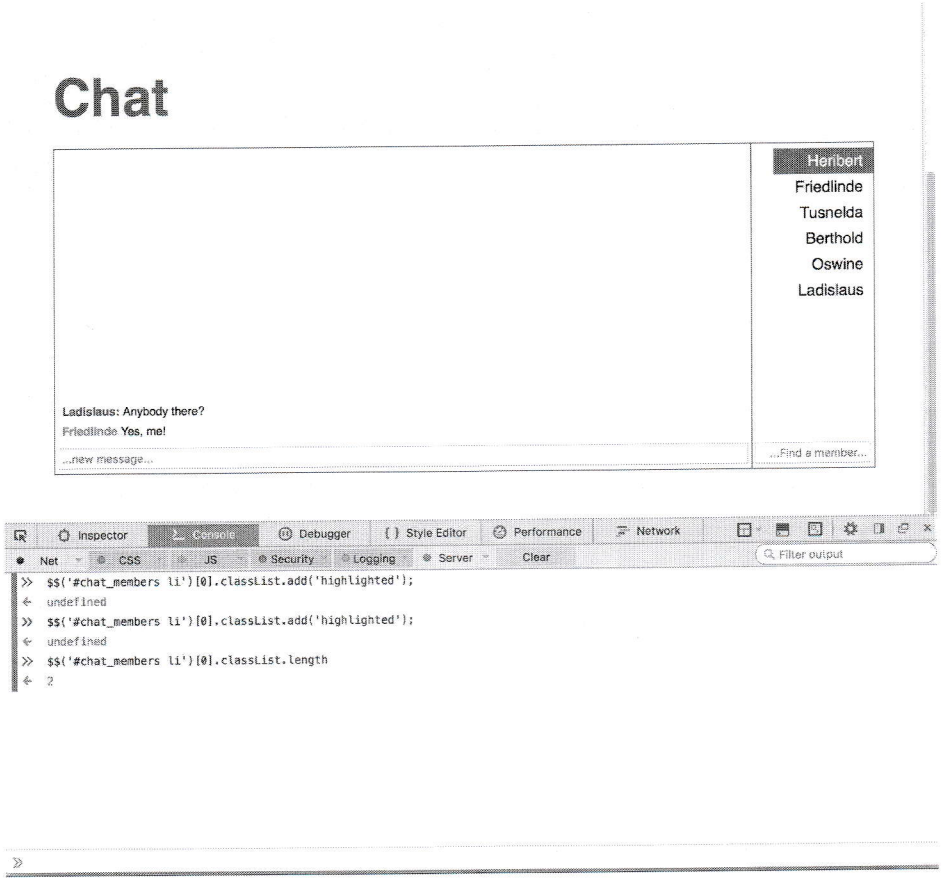
3.2.2 Was zuviel ist, ist zuviel — mit remove

Wenn Sie eine Klasse lieber wieder loswerden möchten, können Sie dazu `remove` verwenden.

```
1 $$('#chat_members li')[0].classList.remove('highlighted');
2 $$('#chat_members li')[0].classList.length // => 1
```

3.2.3 Wechselspiel mit toggle

Die Funktion `toggle` schaltet sogar zwischen Hinzufügen und Entfernen hin und her. Geben Sie folgenden Code mindestens zweimal ein, um den Effekt zu beobachten:



Einen weiteren Unterschied erkennen Sie, wenn Sie eine bereits vorhandene Klasse ein weiteres Mal hinzufügen:

3.2.2 Was zuviel ist, ist zuviel — mit remove

Wenn Sie eine Klasse lieber wieder loswerden möchten, können Sie dazu `remove` verwenden.

```
1 $$('#chat_members li')[0].classList.remove('highlighted');
2 $$('#chat_members li')[0].classList.length // => 1
```

3.2.3 Wechselspiel mit toggle

Die Funktion `toggle` schaltet sogar zwischen Hinzufügen und Entfernen hin und her. Geben Sie folgenden Code mindestens zweimal ein, um den Effekt zu beobachten:

```
$$('#chat_members li')[0].classList.toggle('highlighted');
```

3.2.4 Are you there? — mit contains

Zu guter Letzt können Sie noch prüfen, ob eine bestimmte Klasse vorhanden ist:

```
1 $$('#chat_members li')[0].classList.contains('admin'); // => true
2 $$('#chat_members li')[0].classList.contains('i_am_not_here'); //
=> false
```

3.3 Nur highlighten, was man highlighten will — mit filter

Nun wissen Sie, wie Sie einen Teilnehmer hervorheben können. Sie müssen nur noch herausfinden, **welche** Teilnehmer Sie hervorheben müssen. Tatsächlich müssen Sie aus der Menge aller Teilnehmer erst einmal die *herausfiltern*, die dem eingegebenen Suchstring entsprechen. Die Funktion `filter` könnte Ihnen hier weiterhelfen. Genau wie `forEach` ist auch `filter` nur für Arrays definiert — nicht für NodeLists. Verwenden Sie wieder `Array.from`, um die NodeList in ein Array zu konvertieren.

```
const liNodes = Array.from($$('#chat_members li'));
```

`liNodes` ist nun ein Array, das die `li`-Knoten enthält. Filtern Sie die Knoten heraus, deren Inhalt (`innerHTML`) den gesuchten String (z. B. *ert*) enthält.

```
1 const $ = document.querySelector.bind(document);
2 const $$ = document.querySelectorAll.bind(document);
3 const searchFor = 'ert';
4 const liNodes = Array.from($$('#chat_members li'));
5 const liNodesFound = liNodes.filter(liNode =>
liNode.innerHTML.includes(searchFor));
```

Zuletzt fügen Sie jedem gefundenen Teilnehmer noch die Klasse `highlighted` hinzu:

```
1 const $ = document.querySelector.bind(document);
2 const $$ = document.querySelectorAll.bind(document);
3 const searchFor = 'ert';
4 const liNodes = Array.from($$('#chat_members li'));
5 const liNodesFound = liNodes.filter(liNode =>
6   liNode.innerHTML.includes(searchFor));
7 liNodesFound.forEach(li => li.classList.add('highlighted'));
```

Codebeispiel 3.2 *Highlighten von Mitgliedsnamen, die den String 'ert' enthalten*

Heribert
Friedlinde
Tusnelda
Berthold
Oswine
Ladislaus

3.4 Methoden???

Fall Sie es nicht bemerkt haben, wir hatten gemeinerweise den Begriff ***Methode*** schon eingeschmuggelt, ohne zu erklären, was Methoden eigentlich sind. Wir entschuldigen uns und werden es nun nachholen.

Methoden sind eine spezielle Art von Funktionen. Sie unterscheiden sich von anderen Funktionen dadurch, dass Sie zu einem Objekt gehören. Sie erkennen sie daran, dass dem Funktionsnamen beim Aufruf ein Punkt vorangestellt ist: Beispielsweise bezieht sich die Methode `add` in `li.classList.add("highlighted")` auf die vorhergehende `classList`. Das `add` fügt die Klasse auch nicht zu irgendeiner `classList` hinzu, sondern zu der des entsprechenden `li`-Elements. Insofern können Sie sich das Objekt vor dem Punkt vorstellen wie ein weiteres Argument, das in die Funktion eingeht.

3.5 Zusammenfassung

Methode	Beispiel	Beschreibung
add	$("#bar").classList.add("foo")	fügt eine Klasse hinzu. Im Beispiel erhält das Element mit der `id` *bar* die Klasse *foo*.
remove	$("#bar").classList.remove("foo")	entfernt eine Klasse. Enfernt im Beispiel die Klasse *foo* von Element mit der `id` *bar*.

Tabelle 3.1 *Methoden des classList-Interfaces. Ein ausführliche Referenz finden Sie im MDN*[21]

21. *https://developer.mozilla.org/en/docs/Web/API/Element/classList*

Methode	Beispiel	Beschreibung
item	$("#bar").classList.item(2)	gibt eine Klasse aus der `classList` anhand des übergebenen Index-Arguments zurück. Gibt im Beispiel den dritten Klassennamen des Elements mit der `id` *bar* zurück.
toggle	$("#bar").classList.toggle("foo")	entfernt oder fügt eine Klasse hinzu. Im Beispiel wird die Klasse *foo* hinzugefügt, falls das Element mit der `id` *bar* noch nicht über diese Klasse verfügt. Falls die Klasse schon vorhanden ist, wird sie entfernt.
contains	$("#bar").classList.contains("foo")	gibt `true` zurück, falls die angegebene Klasse in der `classList` vorhanden ist.

Tabelle 3.1 *Methoden des classList-Interfaces. Ein ausführliche Referenz finden Sie im MDN*

Beispiel zur `classList`

HTML-Struktur (vorher):

```
<body><p id="bar" class="foo"><p></body>
```

JavaScript in der Kosole:

```
$("#bar").classList.contains("foo") //true
$("#bar").classList.item(0) // => "foo"
$("#bar").classList.remove("foo")
$("#bar").classList.add("xyz")
```

HTML-Struktur (nachher):

```
<body><p id="bar" class="xyz"><p></body>
```

3.6 Übungen

Übung 7: 01001000010011110101010 0 — Teil 5

Fragen Sie nicht nach dem Sinn, aber nun besteht der Wunsch, den Text teilweise in grauer Schriftfarbe darzustellen.

1. Schreiben Sie eine Funktion, die allen p-Elementen die CSS-Klasse `gray` zuordnet.
2. Eine Ausnahme soll nur der Fließtext darstellen, der den Kaufvorgang enthält. Die p-Elemente mit der Klasse `buy_info_text` sollen dementsprechend von dem Vorgang ausgeschlossen werden.
3. Weisen Sie nun allen Listenpunkten, die noch **keine** andere Klasse haben, die Klasse `gray` zu.

Übung 8: Auf die Länge kommt es an???

Einer Ihrer Kunden wendet sich mit folgendem Wunsch an Sie:

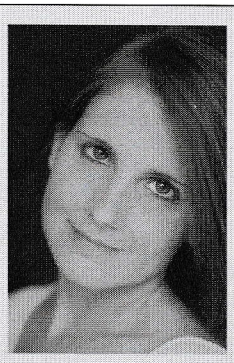
> *Leider sind viele unserer Besucher eher kurz angebunden. Im Forum haben sich nun einige gewünscht, dass die Länge des Artikels schon aus der Überschrift (`h1`) klar hervorgeht. Unser Designer hat sich deswegen drei Stile für Überschriften überlegt und diese auch als entsprechende CSS-Klassen hinterlegt.*

Messen Sie die Länge eines Artikels in Zeichen und geben Sie den Überschriften automatisch die richtige Klasse.

- `coffe_break_article` bis 3000 Zeichen
- `normal_length_article` bis 9000 Zeichen
- `lone_weekend_article` ab 9000 Zeichen

Es geht um die Detailseiten der Artikel. Gehen Sie davon aus, dass jede dieser Seiten nur genau eine `h1`-Überschrift hat und der komplette Artikel (inklusive Bildern und HTML-Struktur) sich in einem Element mit der id *content* befindet.

Für das Zählen der Zeichen dürfen Sie den kompletten HTML-Code von *content* auswerten. Es geht nur darum, eine grobe Abschätzung zu bekommen.



Die Heimtücke der scheinbar unscheinbaren Fehler

Jeder, der schon mal stundenlang verzweifelt einen Fehler gesucht hat, nur um dann festzustellen, dass es sich um einen Syntax-, Tipp,- oder Flüchtigkeitsfehler handelte, weiß, wie ärgerlich das ist. Meine daraus resultierende Erkenntnis, kuriose Fehler betreffend: Sie können ÜBERALL sein! … Naja, vielleicht nicht überall, aber doch an unvermuteten Stellen, und gerade dann führen sie nicht selten zu den merkwürdigsten und unlogischsten Problemen.

Also: Egal ob Sie gerade »nur« HTML & CSS, JS oder sonstigen Code schreiben: sorgfältiges und achtsames Arbeiten zahlt sich immer aus.

4 Und nun: ab ins HTML

Auf der Konsole alle Befehle einzeln einzugeben, wird langsam aber sicher etwas mühsam. Deswegen sollten Sie den Code aus der letzten Lektion nun in einer JavaScript-Datei ablegen.

Schritt für Schritt:

1. Öffnen Sie einen Editor Ihrer Wahl, z.B. Scratchpad.
2. Legen Sie eine neue Datei an, falls nötig. Kopieren Sie den Code aus Codebeispiel 3.2 in diese Datei und speichern Sie sie unter dem Namen *highlight_chat_members.js*
3. Binden Sie die JS-Datei in der HTML-Datei *additional_files/03/examples/chatteilnehmer.html* ein:

```
1 <head>
2   …
3   <script
4     src="highlight_chat_members.js"
5     defer="defer"></script>
6 </head>
```

Sobald Sie die HTML-Datei im Browser öffnen, hebt Ihr JS-Code beim Laden der Seite die passenden Mitglieder hervor.

4. Ändern Sie nun den String `searchFor` von `'ert'` in `'a'`.
5. Führen Sie einen Browser-Reload (**ctrl-R** bzw. **cmd-R**) durch.

Das Script hebt nun andere passende Mitglieder hervor — die, deren Namen `'a'` enthalten.

6. Ändern Sie den String `searchFor` von 'a' wieder in den ursprünglichen Wert `'ert'` zurück und führen Sie erneut einen Reload durch.

4.1 Rauszögern, ohne abzuwarten: Defer & Async

Vielleicht ist Ihnen aufgefallen, dass das `script`-Element zusätzlich mit dem Attribut `defer="defer"` ausgestattet ist. Wofür ist das gut?

Lassen Sie mal versuchsweise das Attribut weg. Sie sehen dann, dass das Hervorheben nicht mehr funktioniert. Die Ursache ist ein Timing-Problem. Damit das Hervorheben funktioniert, müssen die entsprechenden DOM-Elemente (also hier die Chatmitglieder) bereits vorhanden sein, wenn der JS-Code darauf zugreift. Zu dem Zeitpunkt, an dem der Browser das Script lädt (im `head`), sind die Elemente aber noch nicht da. Der Browser findet sie erst später — unten im `body`.

Das Problem ist leicht zu lösen, indem Sie das `script`-Element mit dem `defer`-Attribut ausstatten. `defer` verzögert die Ausführung des Scriptes. Erst, wenn die Seite ihr DOM komplett aufgebaut hat, führt sie das Script aus.

Ein weiterer Vorteil von `defer` besteht darin, dass der Browser das Script *parallel* zum weiteren Parsen der HTML-Seite lädt. Dadurch muss der HTML-Aufbau nicht auf das Laden und Ausführen der JavaScript-Datei warten. Das HTML wird schneller angezeigt.

Ein verwandtes Attribut ist `async`. Es sorgt genauso wie `defer` dafür, dass der Browser die JS-Datei parallel zu anderen Dateien lädt und das Ausführen des HTML-Codes an dieser Stelle nicht warten muss. Im Gegensatz zu `defer` führt der Browser die JS-Datei jedoch sofort aus, sobald sie geladen ist. Es gibt keine Garantie, dass das DOM bereits aufgebaut ist. Außerdem entspricht die Lade- und Ausführungsreihenfolge der JS-Dateien nicht unbedingt der Reihenfolge der script-Tags im HTML-Quellcode (sogenannte ***Source-Order***). Jedes Script startet sobald es geladen ist — d. h. kleinere Scripte starten meistens vor großen.

»Also früher, da mussten wir noch…«

Bevor Browser das `defer`-Attribut »korrekt« unterstützten, waren Entwickler auf Tricks angewiesen. *Korrekt* bedeutet hier, dass der IE vor Version 11 das Attribut zwar unterstützte, aber die Verarbeitungsreihenfolge der Scripte nicht garantiert war. Chrome und Firefox unterstützen das Attribut schon sehr lange spezifikationskonform (Firefox 3.5 & Chrome 8).

Einer dieser Tricks war es beispielsweise, die Scripte nicht im `head` zu platzieren, sondern im `body` — direkt vor dem schließenden body-Tag.

Alle diese Tricks hatten ihre spezifischen Nachteile und/oder waren mit Zusatzaufwand verbunden.

4.2 Strikt und abgeschottet: Blocks & use strict

Nun sollten Sie innerhalb der JS-Datei noch zwei Dinge ergänzen. Zum einen ist es sinnvoll, ein `"use strict";` zu ergänzen, um von der strikteren Syntaxprüfung der JS-Engine zu profitieren — zum anderen sollten Sie den Code komplett in einen Block, d.h. in geschweifte Klammern `{}` packen.

```
1 "use strict";
2 {
3   // your code here
4   …
5 }
```

Ein Block errichtet einen Scope, einen Lebensraum für Variablen. Alle Variablen (`let`) oder Konstanten (`const`), die Sie innerhalb des Blocks anlegen, sind nicht global und können somit nicht mit anderen Bezeichnern in Konflikt treten. Sie wissen nie, ob später noch andere JS-Dateien eingebunden werden, die vielleicht die gleichen Variablennamen verwenden.

»Also früher, da mussten wir noch...«

Früher haben wir eine sogenannte ***IIFE*** verwendet, um zu verhindern, dass sich globale Bezeichner überall ausbreiten. Variablen, die wir mittels `var` angelegt hatten, hatten keinen Block-Scope, wie moderne Variablen (`let`) oder Konstanten (`const`).

Da sich alte Variablen von einem Block nicht begrenzen ließen, musste etwas anderes her. Variablen mit `var` haben Function-Scope — entsprechend lassen Sie sich mit einer Funktion einfangen.

Genau darin besteht der Trick. Wir haben anonyme Funktionen angelegt und diese sofort wieder ausgeführt — nicht, weil wir wirklich eine Funktion gebraucht hätten, sondern nur, um den Gültigkeitsbereich der Variablen einzuschränken:

```
(function() {
  ... Your code here ...
})();
```

IIFE steht für *Immediatly Invoked Function Expression* — ein Funktionsausdruck, der sofort ausgeführt wird.

4.3 Nur ein kleines bisschen Refactoring ...

Hier ist nochmal die komplette Datei zu den vorherigen Beispielen:

```
1  "use strict";
2
3  {
4    const searchFor = "ert";
5
6    const liNodes = Array.from($$("#chat_members li"));
7    const liNodesFound = liNodes.filter(liNode =>
8      liNode.innerHTML.includes(searchFor));
9
10   liNodesFound.forEach(li => li.classList.add("highlighted"));
11
12   const $ = document.querySelector.bind(document);
13   const $$ = document.querySelectorAll.bind(document);
14 }
```

Codebeispiel 4.1 *additional_files/04/examples/highlight_chat_members.js*

Da der Code nun in einer Datei liegt, und Sie ihn nicht mehr direkt in die Konsole tippen, wird die Les- und Wartbarkeit des Codes immer wichtiger. Der Code aus Codebeispiel 4.1 ist nicht gerade ein Musterbeispiel für Clean Code (sauberen, lesbaren Code). Mit ein paar Handgriffen können Sie ihn aber in Form bringen. Das Verbessern von Code —ohne dabei das Verhalten zu ändern — heißt übrigens ***Refactoring*** (Opdyke, Johnson 1990). Es gibt reichlich Literatur zu diesem Thema — z. B. das bekannte Buch *Refactoring* von Martin Fowler (1999).

Der Fokus des Buchs, das Sie gerade in den Händen halten, liegt allerdings nicht auf dem Thema *Refactoring*. Deswegen möchten wir an dieser Stelle nicht zu sehr abschweifen und sämtliche Refactoring-Techniken erläutern. Andererseits haben wir aber den Anspruch, dass Sie lernen, Code zu schreiben, auf den Sie stolz sein können. Deswegen möchten wir Ihnen zumindest an diesem kleinen Beispiel zur Illustration ein paar Verbesserungen zeigen. Den Code aus Codebeispiel 4.1 können wir so wirklich nicht stehen lassen — sonst zeigen Sie ihn all Ihren Bekannten (oder dem JS-Profi von nebenan) und unser Ruf ist endgültig im Eimer.

Also — was können Sie tun, um den Code zu verbessern?

Extract Method

Erst einmal sollte der Code nicht einfach so dastehen. Packen Sie ihn in eine eigene Funktion. Als Funktionsbezeichner bietet sich `highlightChatMembers` an:

```
1  "use strict";
2
3  {
4    const highlightChatMembers = () => {
5      const searchFor = "ert";
6
7      const liNodes = Array.from($$("#chat_members li"));
8      const liNodesFound = liNodes.filter(liNode =>
9        liNode.innerHTML.includes(searchFor));
10
11     liNodesFound.forEach(li => li.classList.add("highlighted"));
12   };
13
14   const $ = document.querySelector.bind(document);
15   const $$ = document.querySelectorAll.bind(document);
16
17   highlightChatMembers();
18 }
```

Codebeispiel 4.2 *additional_files/04/examples/highlight_chat_members_ref1.js*

Statt den gesuchten String in einer Konstante zu hinterlegen, sollten Sie einen Parameter verwenden. Der gesuchte String kommt später vom Anwender! Als Parametername ist `partOfMemberName` im aktuellen Kontext etwas genauer und verständlicher als `searchFor`.

```
1  "use strict";
2
3  {
4    const highlightChatMembersBy = partOfMemberName => {
5      const liNodes = Array.from($$("#chat_members li"));
6      const liNodesFound = liNodes.filter(liNode =>
7        liNode.innerHTML.includes(partOfMemberName));
8
9      liNodesFound.forEach(li => li.classList.add("highlighted"));
10   };
11
12   const $ = document.querySelector.bind(document);
13   const $$ = document.querySelectorAll.bind(document);
14
15   highlightChatMembersBy("ert");
16 }
```

Codebeispiel 4.3 *additional_files/04/examples/highlight_chat_members_ref2.js*

Außerdem war der alte Funktionsname `highlightChatMembers` nun nicht mehr treffend. Der Name `highlightChatMembersBy` verdeutlicht, dass die Funktion einen Parameter (hier: `partOfMemberName`) entgegennimmt. Sie können die Signatur

```
highlightChatMembersBy(partOfMemberName)
```

praktisch wie einen Satz aussprechen:

> *»markiere die Chatteilnehmer anhand eines Teils ihres Namens«*

Ein wichtiges Prinzip beim Programmieren ist das ***Single Responsibility Principle*** (Martin 2002). Angewandt auf Funktionen bedeutet es: Jede Funktion sollte immer nur eine Sache tun. Die Funktion `highlightChatMembersBy` tut eindeutig zu viel: Chat-Mitglieder finden, filtern und hervorheben ... Die richtige Medizin dagegen nennt sich ***Extract Method*** (Fowler 1999) — das Extrahieren neuer Methoden bzw. Funktionen aus bestehendem Code.

```
1  "use strict";
2
3  {
4    const highlightChatMembersBy = partOfMemberName => {
5      const liNodesFound = chatMembers()
6        .filter(liNode =>
7          liNode.innerHTML.includes(partOfMemberName));
8
9      liNodesFound.forEach(highlight);
10   };
11
12   const chatMembers = () => Array.from($$("#chat_members li"));
13   const highlight = el => el.classList.add("highlighted");
14
15   const $ = document.querySelector.bind(document);
```

```
1  "use strict";
2
3  {
4    const highlightChatMembersBy = partOfMemberName => {
5      const liNodes = Array.from($$("#chat_members li"));
6      const liNodesFound = liNodes.filter(liNode =>
7        liNode.innerHTML.includes(partOfMemberName));
8
9      liNodesFound.forEach(li => li.classList.add("highlighted"));
10   };
11
12   const $ = document.querySelector.bind(document);
13   const $$ = document.querySelectorAll.bind(document);
14
15   highlightChatMembersBy("ert");
16 }
```

Codebeispiel 4.3 *additional_files/04/examples/highlight_chat_members_ref2.js*

Außerdem war der alte Funktionsname `highlightChatMembers` nun nicht mehr treffend. Der Name `highlightChatMembersBy` verdeutlicht, dass die Funktion einen Parameter (hier: `partOfMemberName`) entgegennimmt. Sie können die Signatur

```
highlightChatMembersBy(partOfMemberName)
```

praktisch wie einen Satz aussprechen:

> *»markiere die Chatteilnehmer anhand eines Teils ihres Namens«*

Ein wichtiges Prinzip beim Programmieren ist das ***Single Responsibility Principle*** (Martin 2002). Angewandt auf Funktionen bedeutet es: Jede Funktion sollte immer nur eine Sache tun. Die Funktion `highlightChatMembersBy` tut eindeutig zu viel: Chat-Mitglieder finden, filtern und hervorheben ... Die richtige Medizin dagegen nennt sich ***Extract Method*** (Fowler 1999) — das Extrahieren neuer Methoden bzw. Funktionen aus bestehendem Code.

```
1  "use strict";
2
3  {
4    const highlightChatMembersBy = partOfMemberName => {
5      const liNodesFound = chatMembers()
6        .filter(liNode =>
7          liNode.innerHTML.includes(partOfMemberName));
8
9      liNodesFound.forEach(highlight);
10   };
11
12   const chatMembers = () => Array.from($$("#chat_members li"));
13   const highlight = el => el.classList.add("highlighted");
14
15   const $ = document.querySelector.bind(document);
```

```
16   const $$ = document.querySelectorAll.bind(document);
17
18   highlightChatMembersBy("ert");
19 }
```

Codebeispiel 4.4 *additional_files/04/examples/highlight_chat_members_ref3.js*

Das Finden der Mitglieder übernimmt nun die Funktion `chatMembers`, während `highlight` sich um die Hervorhebung (engl. *Highlighting*) eines einzelnen Elements kümmert. Ein weiterer Kandidat für eine eigene Funktion ist das Prüfen auf Treffer:

```
liNode.innerHTML.includes(partOfMemberName)
```

Ein geeigneter Bezeichner für die Funktion wäre `doesMemberMatch`. Codebeispiel 4.5 zeigt den neuen Code mit extrahierten Funktionen.

```
1 "use strict";
2
3 {
4   const highlightChatMembersBy = partOfMemberName => {
5     const liNodesFound = chatMembers()
6       .filter(memberElement =>
7         doesMemberMatch(partOfMemberName, memberElement));
8
9     liNodesFound.forEach(highlight);
10  };
11
12  const doesMemberMatch = (partOfMemberName, memberElement) =>
13    memberElement.innerHTML.includes(partOfMemberName);
14
15  const chatMembers = () => Array.from($$("#chat_members li"));
16  const highlight = el => el.classList.add("highlighted");
17
18  const $ = document.querySelector.bind(document);
19  const $$ = document.querySelectorAll.bind(document);
20
21  highlightChatMembersBy("ert");
22 }
```

Codebeispiel 4.5 *additional_files/04/examples/highlight_chat_members_ref4.js*

Jetzt fällt vielleicht auf, dass Sie die Konstante `liNodesFound` im Grunde nicht mehr benötigen. Auch der Name *liNodesFound* trägt nicht unbedingt dazu bei, den Code verständlicher zu machen. Nehmen Sie die Bezeichner einfach raus. Damit verkürzen Sie:

```
1 …
2 const liNodesFound = chatMembers()
3   .filter(memberElement =>
4     doesMemberMatch(partOfMemberName, memberElement));
5 liNodesFound.forEach(highlight);
6 …
```

auf eine Zeile (auch wenn wir sie aus Lesbarkeitsgründen in mehrere Zeilen umbrechen):

```
1 …
2 chatMembers()
3   .filter(memberElement =>
4     doesMemberMatch(partOfMemberName, memberElement))
5   .forEach(highlight);
6 …
```

Damit ist die Implementierung der Funktion `highlightChatMembersBy` auch leicht zu lesen:

1. Besorge die chatMembers.
2. Filtere die zu `partOfMemberName` passenden (engl. *does match*) heraus.
3. Hebe alle gefundenen hervor.

Wer die Details verstehen möchte, kann die einzelnen Funktionen jederzeit nachlesen, aber so kommuniziert `hightlightChatMembersBy` schon mal die groben Schritte, die vorher in den Details versteckt waren.

Newspaper-Metapher

Abb. 4.1 *Foto: Olu Eletu*[22]*, CC0 1.0*[23]

22. *https://unsplash.com/photos/E7RLgUjjazc*
23. *http://creativecommons.org/publicdomain/zero/1.0/*

Als Nächstes bietet es sich an, die neuen, extrahierten Funktionen etwas zu sortieren: Grobe High-Level-Funktionen gehören nach oben, detaillierte Low-Level-Hilfsfunktionen nach unten. Dadurch können Sie Ihren Code lesen wie eine Zeitung — oben stehen die »dicken« Headlines, und wer mehr wissen möchte, erfährt sukzessive mehr Details. Das ist die sogenannte ***Newspaper-Metapher*** von Robert C. Martin (2008).

```
1  "use strict";
2
3  {
4    const highlightChatMembersBy = partOfMemberName => {
5      chatMembers()
6        .filter(member =>
7          doesMemberMatch(partOfMemberName, member))
8        .forEach(highlight);
9    };
10
11   const doesMemberMatch = (partOfMemberName, memberElement) =>
12     memberElement.innerHTML.includes(partOfMemberName);
13
14   const chatMembers = () => Array.from($$("#chat_members li"));
15   const highlight = el => el.classList.add("highlighted");
16
17
18   const $ = document.querySelector.bind(document);
19   const $$ = document.querySelectorAll.bind(document);
20
21   highlightChatMembersBy("ert");
22 }
```

Codebeispiel 4.6 *additional_files/04/examples/highlight_chat_members_ref5.js*

4.4 Array.from war gestern. Jetzt heißt es: Array-Methoden für alle!

Jedes Mal, wenn Sie auf dem Ergebnis der `$$`-Funktion eine High-Order-Function wie `map` oder `filter` aufrufen möchten, müssen Sie es vorher mit `Array.from` konvertieren, z. B.

```
1 const chatMembers = () => Array.from($$("#chat_members li"));
```

Finden Sie das nicht auch ein wenig nervig?

Abhilfe bietet die folgende Zeile (aus dem legendären *bling.js-Gist von Paul Irish*[24]). Sie müssen sie nicht im Detail verstehen, aber sie sorgt im Grunde dafür, dass einer *NodeList* auch alle Methoden eines `Arrays` zur Verfügung stehen.

24. *https://gist.github.com/paulirish/12fb951a8b893a454b32*

```
NodeList.prototype.__proto__ = Array.prototype;
```

Damit entfällt die Notwendigkeit, die *NodeList* mittels `Array.from` in ein `Array` zu wandeln. Die Funktion `chatMembers` wird dadurch recht einfach.

```
1  "use strict";
2
3  {
4    const highlightChatMembersBy = partOfMemberName => {
5      chatMembers()
6        .filter(member =>
7          doesMemberMatch(partOfMemberName, member))
8        .forEach(highlight);
9    };
10
11   const doesMemberMatch = (partOfMemberName, memberElement) =>
12     memberElement.innerHTML.includes(partOfMemberName);
13
14   const chatMembers = () => $$("#chat_members li");
15   const highlight = el => el.classList.add("highlighted");
16
17   const $ = document.querySelector.bind(document);
18   const $$ = document.querySelectorAll.bind(document);
19   NodeList.prototype.__proto__ = Array.prototype;
20
21   highlightChatMembersBy("ert");
22 }
```

Codebeispiel 4.7 *additional_files/04/examples/highlight_chat_members_ref6.js*

Zugegeben: Hier lohnt es sich kaum, da das `Array.from` nur ein einziges Mal im Code vorkommt. Es macht aber durchaus Sinn, die Zeile einfach einzubinden, um dann etwas entspannter zu programmieren. Sie müssen dann nicht jedes Mal darüber nachdenken, ob Sie vielleicht gerade eine Methode von `Array` auf der *NodeList* verwenden und deswegen erst konvertieren müssen.

Aufpassen müssen Sie nur, wenn Sie weitere Drittbibliotheken einbinden, die evtl. ebenfalls die *NodeList* verändern könnten. In diesem Fall müssen Sie eben wieder zu `Array.from` zurück.

4.5 Und jetzt noch eine klitzekleine Verbesserung

Leider gibt es auch noch einen kleinen Bug in der Anwendung, den Sie sich nun. Wenn Sie als Suchstring `'Bert'` verwenden, wird nur Berthold, aber nicht Heribert selektiert. Das liegt daran, dass der Stringvergleich ***case sensitive*** ist, also Groß- und Kleinschreibung berücksichtigt. Eine einfache Möglichkeit, Groß- und Kleinschreibung bei der Suche zu ignorieren, ist, beide am Vergleich beteiligten Strings vorher komplett in Kleinbuchstaben umzuwandeln — mit der String-Methode `toLowerCase`.

```
NodeList.prototype.__proto__ = Array.prototype;
```

Damit entfällt die Notwendigkeit, die *NodeList* mittels `Array.from` in ein `Array` zu wandeln. Die Funktion `chatMembers` wird dadurch recht einfach.

```
1  "use strict";
2
3  {
4    const highlightChatMembersBy = partOfMemberName => {
5      chatMembers()
6        .filter(member =>
7          doesMemberMatch(partOfMemberName, member))
8        .forEach(highlight);
9    };
10
11   const doesMemberMatch = (partOfMemberName, memberElement) =>
12     memberElement.innerHTML.includes(partOfMemberName);
13
14   const chatMembers = () => $$("#chat_members li");
15   const highlight = el => el.classList.add("highlighted");
16
17   const $ = document.querySelector.bind(document);
18   const $$ = document.querySelectorAll.bind(document);
19   NodeList.prototype.__proto__ = Array.prototype;
20
21   highlightChatMembersBy("ert");
22 }
```

Codebeispiel 4.7 *additional_files/04/examples/highlight_chat_members_ref6.js*

Zugegeben: Hier lohnt es sich kaum, da das `Array.from` nur ein einziges Mal im Code vorkommt. Es macht aber durchaus Sinn, die Zeile einfach einzubinden, um dann etwas entspannter zu programmieren. Sie müssen dann nicht jedes Mal darüber nachdenken, ob Sie vielleicht gerade eine Methode von `Array` auf der *NodeList* verwenden und deswegen erst konvertieren müssen.

Aufpassen müssen Sie nur, wenn Sie weitere Drittbibliotheken einbinden, die evtl. ebenfalls die *NodeList* verändern könnten. In diesem Fall müssen Sie eben wieder zu `Array.from` zurück.

4.5 Und jetzt noch eine klitzekleine Verbesserung

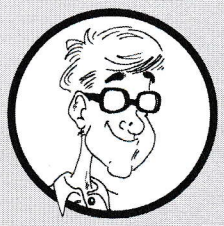
Leider gibt es auch noch einen kleinen Bug in der Anwendung, den Sie sich nun. Wenn Sie als Suchstring `'Bert'` verwenden, wird nur Berthold, aber nicht Heribert selektiert. Das liegt daran, dass der Stringvergleich ***case sensitive*** ist, also Groß- und Kleinschreibung berücksichtigt. Eine einfache Möglichkeit, Groß- und Kleinschreibung bei der Suche zu ignorieren, ist, beide am Vergleich beteiligten Strings vorher komplett in Kleinbuchstaben umzuwandeln — mit der String-Methode `toLowerCase`.

```
1  "use strict";
2
3  {
4    const highlightChatMembersBy = partOfMemberName => {
5      chatMembers()
6        .filter(member =>
7          doesMemberMatch(partOfMemberName, member))
8        .forEach(highlight);
9    };
10
11   const doesMemberMatch = (partOfMemberName, memberElement) =>
12     memberElement.innerHTML.toLowerCase()
13       .includes(partOfMemberName.toLowerCase());
14
15   const chatMembers = () => $$("#chat_members li");
16   const highlight = el => el.classList.add("highlighted");
17
18   const $ = document.querySelector.bind(document);
19   const $$ = document.querySelectorAll.bind(document);
20   NodeList.prototype.__proto__ = Array.prototype;
21
22   highlightChatMembersBy("ert");
23 }
```

Codebeispiel 4.8 *additional_files/04/examples/highlight_chat_members_ref6_lowercase.js*

Es gäbe übrigens noch genügend Möglichkeiten, das Refactoring fortzuführen, etwa mit Hilfe von ***funktionaler*** oder ***objektorientierter Programmierung*** oder gar ***hexagonaler Architektur***. »Doch das ist eine andere Geschichte und soll ein anderes Mal erzählt werden.«

Events feiern, wie sie fallen … 5



Wenn das neue Chat-Feature heute Abend fertig ist, veranstalten wir eine Release-Party im Büro. Nichts Großes — aber es gibt was zu trinken, gute Musik und auch eine Kleinigkeit zu essen …

Abb. 5.1 *Foto: Yutacar*[25]*, CC0 1.0*[26]

Der Code ist nun in eine JavaScript-Datei ausgelagert und etwas »sauberer« geworden. Leider ist Björn noch nicht zufrieden. Da fehlt noch was…

Natürlich sollten Besucher der Website kein JS programmieren müssen, um Chatmitglieder hervorzuheben. Die Hervorhebungen sollten sich automatisch anhand der Eingabe im Suchfeld ändern.

Dazu muss der Code auf Benutzereingaben reagieren. Der Browser stellt dazu sogenannte ***Events*** zur Verfügung. Mit Events sind hier also leider keine Hochzeiten oder Partys gemeint.

5.1 Events im Browser

Für den Browser ist schon jede kleine Einflussnahme durch den Anwender ein Event. Ein Event kann dabei fast alles sein, beispielsweise:

25. *https://unsplash.com/photos/JKMnm3CIncw*
26. *http://creativecommons.org/publicdomain/zero/1.0/*

- das Anklicken eines Buttons
- das Überfahren eines Bildes mit der Maus
- das Loslassen einer Taste
- das Verlassen eines Eingabefeldes
- Fingergesten auf Touchbildschirmen, wie z. B. Zoom

Es gibt auch Events, die nicht direkt vom Anwender ausgelöst werden, z. B. durch Netzwerkanfragen oder bei abgeschlossenen Ladevorgängen von Dateien. Vorerst konzentrieren wir uns aber auf Events, die unmittelbar mit dem Anwenderverhalten zu tun haben.

5.2 Zum anständigen Behandeln von Events: Eventhandler

5.2.1 Vorbereitende Maßnahmen

Starten Sie zunächst mit einem kleinen Experiment in der Konsole. Das Event, das Sie interessiert, ist das Loslassen einer Taste im Eingabefeld. Im HTML-Dokument des Chats finden Sie das Eingabefeld (`input`-Element) innerhalb eines `div`-Elements mit der `id` *member_search*:

```
1 …
2 <div id="member_search"><input type="text" placeholder="...Find a
member..."/></div>
3 …
```

Selektieren Sie das Feld zunächst mittels `$('#member_search input')`. Sie erhalten das `input`-Element als Rückgabewert in der Konsole. Wenn Sie den Mauszeiger darüber bewegen, erkennen Sie, dass es sich um das richtige Feld handelt.

An dieses Feld müssen Sie ein Event binden. Verwenden Sie dazu die Methode `addEventListener`. Nahezu jedes HTML-Element-Objekt verfügt über diese Methode.

```
$('#member_search input').addEventListener( … );
```

`addEventListener` benötigt zwei Argumente. Das erste Argument ist der ***Eventtyp*** als String. In diesem Fall sollten Sie `'keyup'` verwenden — das Loslassen einer Taste.

```
$('#member_search input').addEventListener('keyup', …);
```

Das zweite Argument muss eine Funktion sein — der Browser führt sie beim Auftreten des Events aus. Verwenden Sie erst einmal etwas Einfaches, um zu sehen, ob es überhaupt funktioniert:

```
() => alert(1)
```

Die Funktion, die Sie auf das Event registrieren, wird übrigens als ***Eventhandler*** bezeichnet. Diese Funktion verarbeitet das angegebene Event.

Damit ist die Anweisung vollständig — geben Sie sie in die Konsole ein:

```
$('#member_search input').addEventListener('keyup', () =>
alert(1));
```

Sobald Sie in das Eingabefeld klicken und eine Taste drücken, erscheint beim Loslassen eine Alert-Box mit der `1`. Das ist sicherlich noch nicht sehr sinnvoll. Vielleicht ist auch das Wegklicken der Alert-Boxen ein wenig nervig, aber Sie wissen nun, dass es funktioniert. Check. Mission erfolgreich.

5.2.2 ... und nun alles zusammen

Nun kommt der etwas schwierigere Teil: Sie müssen aus dem vorhanden Code für Highlighting und Eventregistierung ein vollständiges Programm bauen.

Entfernen Sie den Aufruf `hightlightChatMembers('ert');` aus der JS-Datei. Stattdessen soll das `keyup`-Event den Aufruf der Funktion auslösen:

```
1 $('#member_search input')
2   .addEventListener('keyup', () =>
3     hightlightChatMembers('ert'));
```

...und tatsächlich: Sobald Sie im Suchfeld eine Taste drücken, ändert sich das Highlighting. Statt des festen Highlightings zum String `'ert'` möchten Sie aber die tatsächliche Eingabe verwenden. Wie kommen Sie da dran?

Sie benötigen nochmal das Objekt des Input-Elements. Sie können es sich wieder mittels $('#member_search input') besorgen — und es dann nach dem Wert fragen, den der Benutzer eingegeben hat. Den Wert finden Sie in der Eigenschaft `value` — also `$('#member_search input').value`. Mehr zu dieser und anderen Eigenschaften erfahren Sie in Lektion 7.

Zusammengesetzt erhalten Sie folgende Anweisung:

```
1 $('#member_search input')
2   .addEventListener('keyup', () =>
3     hightlightChatMembers($('#member_search input').value));
```

Damit ergibt sich nun insgesamt folgender Code:

```
1  "use strict";
2
3  {
4    const highlightChatMembersBy = partOfMemberName => {
5      chatMembers()
6        .filter(member =>
7          doesMemberMatch(partOfMemberName, member))
8        .forEach(highlight);
9    };
10
11   const doesMemberMatch = (partOfMemberName, memberElement) =>
12     memberElement.innerHTML.toLowerCase()
13       .includes(partOfMemberName.toLowerCase());
14
15   const chatMembers = () => $$("#chat_members li");
16   const highlight = el => el.classList.add("highlighted");
17
18   const $ = document.querySelector.bind(document);
19   const $$ = document.querySelectorAll.bind(document);
20   NodeList.prototype.__proto__ = Array.prototype;
21
22   $("#member_search input")
23     .addEventListener("keyup", () =>
24       highlightChatMembersBy($("#member_search input").value));
25 }
```

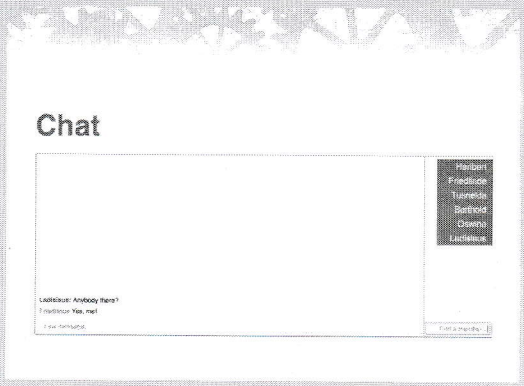
Codebeispiel 5.1 *additional_files/05/examples/highlight_chat_members_1_event/highlight_chat_members.js*

Experimentieren Sie ein wenig mit der aktuellen Implementierung. Geben Sie verschiedene Strings ein. Was fällt Ihnen auf?

5.3 Guard Clauses: Wächter für Ihre Funktionen

Huch — ich hab' gerade einen Bug gefunden. Wenn ich nach Chatmitgliedern suche, passiert es sehr schnell, dass *alle* hervorgehoben werden.

Bitte beheben Sie das! So können wir das Feature unmöglich online stellen.



Tja — das Problem besteht darin, dass das Programm zwar die passenden Mitglieder hervorhebt, aber diese Hervorhebung nicht wieder entfernt.

Statt nun mühsam herauszufinden, welche Hervorhebungen Sie wieder entfernen müssen, ist es einfacher, Sie entfernen einfach alle Hervorhebungen und heben anschließend nur die jeweils passenden hervor. Dazu benötigen Sie weiteren Code, der die CSS-Klasse von allen Mitgliedern entfernt:

```
1 const removeHighlightsFromAllChatMembers = () =>
2   chatMembers().forEach(removeHighlight);
3 const removeHighlight = el => el.classList.remove('highlighted');
```

Im Übrigen ist es überhaupt kein Problem, dass einige der `li`-Elemente mit Chatmitgliedern die Klasse *highlighted* gar nicht haben — in dem Fall wird auch nichts entfernt.

Ordnen Sie die beiden Funktionen passend zu ihrem Detailgrad der Funktionen ein. Außerdem muss die neue Funktion `removeHighlightsFromAllChatMembers` noch aufgerufen werden. Dazu empfiehlt es sich, eine übergeordnete Funktion namens `updateHighlightingOfChatMembers` zu erstellen, die zunächst alle Hight-

lights entfernt (`removeHighlightsFromAllChatMembers`) und danach die richtigen Chatmitglieder markiert (`hightlightChatMembersBy`):

```
1 const updateHighlightingOfChatMembers = partOfMemberName => {
2   removeHighlightsFromAllChatMembers();
3   hightlightChatMembersBy(partOfMemberName);
4 };
```

Nun müssen Sie nur noch die neue Funktion `updateHighlightingOfChatMembers` statt der reinen Markierungsfunktion `hightlightChatMembersBy` in den Eventhandler einbinden. Sie erhalten diesen Code:

```
1 "use strict";
2
3 {
4   const updateHighlightingOfChatMembers = partOfMemberName => {
5     removeHighlightsFromAllChatMembers();
6     highlightChatMembersBy(partOfMemberName);
7   };
8
9   const removeHighlightsFromAllChatMembers = () =>
10    chatMembers().forEach(removeHighlight);
11
12  const highlightChatMembersBy = partOfMemberName => {
13    chatMembers()
14      .filter(member =>
15        doesMemberMatch(partOfMemberName, member))
16      .forEach(highlight);
17  };
18
19  const doesMemberMatch = (partOfMemberName, memberElement) =>
20    memberElement.innerHTML.toLowerCase()
21      .includes(partOfMemberName.toLowerCase());
22
23  const chatMembers = () => $$("#chat_members li");
24  const highlight = el => el.classList.add("highlighted");
25  const removeHighlight = el => el.classList.remove("highlighted");
26
27  const $ = document.querySelector.bind(document);
28  const $$ = document.querySelectorAll.bind(document);
29  NodeList.prototype.__proto__ = Array.prototype;
30
31  $("#member_search input")
32    .addEventListener("keyup", () =>
33      updateHighlightingOfChatMembers($("#member_search
input").value));
34 }
```

Codebeispiel 5.2 *additional_files/05/examples/highlight_chat_members_2_remove/highlight_chat_members.js*

Der Code aus Codebeispiel 5.2 funktioniert … fast. Er funktioniert bis auf die kleine Unschönheit, dass bei einer leeren Eingabe alle Mitglieder selektiert sind. Der leere String kommt in jedem String vor!

Dieses Verhalten können Sie einfach dadurch unterdrücken, dass Sie zu Beginn der Funktion `hightlightChatMembersBy` prüfen, ob `partOfMemberName` leer ist. Bei einem Leerstring muss die Funktion einfach die Arbeit verweigern. Beenden Sie die Funktion vorzeitig mittels `return`.

```
if (partOfMemberName === "") return;
```

Zeilen wie diese, die vor Ausführung einer Funktion prüfen, ob die Werte der eingehenden Argumente überhaupt Sinn machen, heißen *Guard-Clauses*[27] oder einfach ***Guards***. Sie schützen die Funktionen vor ungültigen Eingangswerten.

```
1 const hightlightChatMembersBy = partOfMemberName => {
2   if (partOfMemberName === "") return;
3   chatMembers()
4     .filter(member =>
5       doesMemberMatch(partOfMemberName, member))
6     .forEach(highlight);
7 };
```

Codebeispiel 5.3 *Die Funktion highlightChatMembersBy mit Guard*

Der Code aus Codebeispiel 5.3 verhält sich endlich wie gewünscht. Bevor Sie zu Björn stürmen, um ihm die gute Nachricht zu überbringen, könnten Sie sich vielleicht noch ein wenig Zeit nehmen, um den Code zu verbessern. Ihr Wartungsprogrammierer wird es Ihnen irgendwann danken!

5.4 Der gekonnte Einstieg — mit init

Der Code zur Eventregistierung steht auch noch etwas unmotiviert da. Besser wäre es, ihn in eine Funktion zu packen. Das hat viele Vorteile: Er wird wiederverwendbar und bekommt einen Namen (damit ist er leichter zu identifizieren). Außerdem gibt es weitere Vorteile in Bereichen, mit denen wir uns noch nicht beschäftigt haben — z. B. ist der Code leicher zu testen.

Zwei Funktionsnamen bieten sich hier an: *registerEvents* oder einfach *init*. Wenn Sie möchten, könnten Sie auch noch konkreter werden und die Funktion sogar *registerEventsForChatMemberHighlighting* nennen. In diesem Kontext, in dem die Datei aber nur eine einzige Aufgabe hat, ist das generische *init* durchaus akzeptabel.

Der Name *init* steht für Initialisierung. Das ist quasi die Funktion, die alles andere einleitet und somit einen Einstiegspunkt in den restlichen Code bietet. Im Grunde können Sie die Funktion nennen, wie Sie möchten, aber *init* hat sich etabliert.

27. *http://c2.com/cgi/wiki?GuardClause*

Andere Programmierer lesen den Funktionsnamen und verstehen sofort, was Sie damit meinen.

Im Sinne der Newspaper-Metapher gehört die Funktionsdefinition von `init` an den Anfang. Der Aufruf `init()` kann aber erst erfolgen, wenn alle Definitionen abgeschlossen sind.

```
1 "use strict";
2
3 {
4   const init = () => $("#member_search input")
5     .addEventListener("keyup", () =>
6       updateHighlightingOfChatMembers($("#member_search
input").value));
7
8   const updateHighlightingOfChatMembers = partOfMemberName => {
9     removeHighlightsFromAllChatMembers();
10    hightlightChatMembersBy(partOfMemberName);
11  };
12
13  const removeHighlightsFromAllChatMembers = () =>
14    chatMembers().forEach(removeHighlight);
15
16  const hightlightChatMembersBy = partOfMemberName => {
17    if (partOfMemberName === "") return;
18    chatMembers()
19      .filter(member =>
20        doesMemberMatch(partOfMemberName, member))
21      .forEach(highlight);
22  };
23
24  const doesMemberMatch = (partOfMemberName, memberElement) =>
25    memberElement.innerHTML.toLowerCase()
26      .includes(partOfMemberName.toLowerCase());
27
28  const chatMembers = () => $$("#chat_members li");
29  const highlight = el => el.classList.add("highlighted");
30  const removeHighlight = el => el.classList.remove("highlighted");
31
32  const $ = document.querySelector.bind(document);
33  const $$ = document.querySelectorAll.bind(document);
34  NodeList.prototype.__proto__ = Array.prototype;
35
36  init();
37 }
```

Codebeispiel 5.4 *additional_files/05/examples/highlight_chat_members_3_init/highlight_chat_members.js*

5.5 Schauen Sie hinter die Fassade — mit dem Event-Objekt

Dass Sie hier noch etwas verbessern können wird deutlich, wenn Sie sich die Eventregistrierung genauer betrachten.

```
1 $('#member_search input')
2   .addEventListener('keyup', () =>
3     hightlightChatMembers($('#member_search input').value));
```

Die Funktion, die Sie hier als Eventhandler registrieren, ist diese:

```
() => hightlightChatMembers($('#member_search input').value);
```

Das Interessante dabei ist, dass die Funktion beim Aufruf durch das Event dieses automatisch als Argument übergeben bekommt. Sie müssen der Funktion nur noch einen Parameter spendieren, um das Event aufzufangen:

```
event => hightlightChatMembers($('#member_search input').value);
```

Was können Sie mit dem `event` anfangen? Sie können es beispielsweise loggen, um zu sehen, was es alles beinhaltet:

```
1 event => {
2   console.log(event);
3   hightlightChatMembers($('#member_search input').value);
4 }
```

Wenn Sie die Taste **a** drücken, zeigt die Konsole:

```
keyup { target: <input>, key: "a", charCode: 0, keyCode: 65 }
```

Ein Klick auf das *keyup* verrät Ihnen, dass es sich bei `event` um ein JS-Objekt vom »Typ«[28] `KeyboardEvent` handelt. Keyboardevents besitzen viele interessante Eigenschaften. Einige davon zeigt die folgende Tabelle:

Eigenschaft	Inhalt
type	keyup
code	KeyA
key	a
keyCode	65

28. Eigentlich gibt es in JS keine spezifischen Objekttypen. Korrekt wäre: Es handelt sich um ein JS-Objekt, das das Objekt `KeyboardEvent` in seiner Prototype-Chain hat. Wir verwenden den Begriff *Typ* hier als Vereinfachung. Es geht nur darum, dass alle Objekte des gleichen »Typs« auch die gleichen Eigenschaften haben.

timestamp	[Timestamp, wann das Event sich ereignet hat]
target	[Element, das das Event ausgelöst hat]
ctrlKey	false [war die Ctrl-Taste dabei gedrückt?]
altKey	false [war die Alt-Taste dabei gedrückt?]

Es gibt noch viel mehr Eigenschaften, doch schon diese wenigen verdeutlichen, dass Sie im Eventobjekt alle Informationen finden, die beim Auftreten des Events eine Rolle spielen, z. B.:

- Welche Art von Event ist das? *keyup*
- Welche Taste wurde losgelassen? *a*
- Wann wurde das Event ausgelöst?
- Wo wurde das Event ausgelöst?

Diese Informationen können Sie dann in Ihrem Eventhandler verwerten. Für das aktuelle Programm ist vor allem die Eigenschaft `target` interessant. Da `target` das gesuchte Input-Field enthält, das Sie sonst mittels `$('#member_search input')` selektieren, können Sie den Eventhandler auch folgendermaßen registrieren:

```
event => hightlightChatMembers(event.target.value);
```

Das hat wieder eine Reihe von handfesten Vorteilen. Es ist in jedem Fall performanter, da der Browser das Element-Objekt nicht noch einmal ermitteln muss. In `target` liegt es bereits vor. Der viel wichtigere Vorteil ist allerdings die verbesserte Wartbarkeit. Sollte sich der Selektor zum Finden des Feldes ändern (z. B. weil sich die HTML-Struktur ändert), so müssen Sie nicht daran denken, den Eventhandler anzupassen. Er bezieht sich immer auf das jeweilige `target`, völlig unabhängig davon, wie Sie das Element gefunden und das Event darauf registriert haben.

```
1  "use strict";
2
3  {
4    const init = () => $("#member_search input")
5      .addEventListener("keyup", event =>
6        updateHighlightingOfChatMembers(event.target.value));
7
8    const updateHighlightingOfChatMembers = partOfMemberName => {
9      removeHighlightsFromAllChatMembers();
10     hightlightChatMembersBy(partOfMemberName);
11   };
12
13   const removeHighlightsFromAllChatMembers = () =>
```

```
14      chatMembers().forEach(removeHighlight);
15
16    const hightlightChatMembersBy = partOfMemberName => {
17      if (partOfMemberName === "") return;
18      chatMembers()
19        .filter(member =>
20          doesMemberMatch(partOfMemberName, member))
21        .forEach(highlight);
22    };
23
24    const doesMemberMatch = (partOfMemberName, memberElement) =>
25      memberElement.innerHTML.toLowerCase()
26        .includes(partOfMemberName.toLowerCase());
27
28    const chatMembers = () => $$("#chat_members li");
29    const highlight = el => el.classList.add("highlighted");
30    const removeHighlight = el => el.classList.remove("highlighted");
31
32    const $ = document.querySelector.bind(document);
33    const $$ = document.querySelectorAll.bind(document);
34    NodeList.prototype.__proto__ = Array.prototype;
35
36    init();
37  }
```

Codebeispiel 5.5 *additional_files/05/examples/highlight_chat_members_4_target/highlight_chat_members.js*

Übrigens können Sie `event` noch als `e` abkürzen. Normalerweise gelten Abkürzungen als »böse« und sollten vermieden werden — in diesem Fall geht das aber in Ordnung. Der einzelne Buchstabe `e` ist eine gängige Abkürzung für `event` und ist akzeptabel, solange der Scope (Sichtbarkeitsbereich der Variable) auf eine sehr kurze Funktion begrenzt ist.

Jetzt ist es soweit — Sie können Björn das verbesserte Feature zeigen, und weil er so zufrieden damit ist, schmeißt er nun doch noch eine Party — hoffentlich wird das ein unvergessliches Event!

Danke! Genauso hab' ich mir das vorgestellt! Und hey, wir sehen uns heute Abend!

Übung 9: Please Click Me

1. Der folgende Button soll beim Betätigen eine Alert-Box öffnen mit dem Text »*Hey I like it when you click me!*«.

```
<button>Click Me</button>
```

2. Entfernen Sie die Alert-Box. Beim Betätigen des Buttons soll sich stattdessen der Button-Text ändern.
3. Ändern Sie den Text in »*Cool, you found an easteregg*«, für den Fall, dass der Anwender beim Klicken die Taste **ctrl** bzw. **cmd** gedrückt hält.

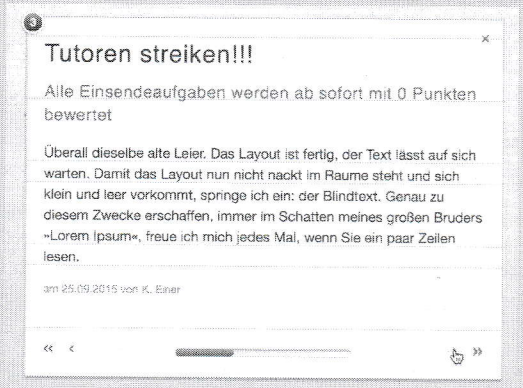
5.6 preventDefault oder: Wer mag schon immer nur »Standard«?

Am übernächsten Tag, als sich Björn wieder ein wenig von der Party erholt hat, präsentiert er die nächste Anforderung.

Wie Sie bestimmt noch wissen, gibt es auf der Startseite einen kleinen Newsbereich. Der Bereich zeigt die aktuellen News in einer Box mit Scrollbalken an. Das möchten wir gerne ändern.

In Zukunft soll dort immer nur eine Nachricht angezeigt werden. Stattdessen soll es dann Vor- und Zurück-Buttons geben, mit denen sich die News durchblättern lassen.

Unser Designer hat das Layout bereits fertig, es fehlt nur noch die Funktionalität.



Wenn ein Kunde »*nur noch*« sagt, ist das meistens kein gutes Zeichen. Na gut, los geht's — schließlich zahlt Björn nicht schlecht…

Später einmal, wenn alles fertig ist, wird der Server die Nachrichten liefern. Damit Sie aber jetzt schon Daten zum Ausprobieren haben, haben wir uns erlaubt, Ihnen Blindtext in Form eines Arrays zur Verfügung zu stellen:

```
const messages = [
  `<h1>Tutoren streiken!!!</h1>
  <h2>Alle Einsendeaufgaben werden ab sofort mit 0 Punkten
bewertet</h2>
  <p>[&hellip;]</p>
  <p>am 25.09.2015 von K. Einer</p>`,
  `<h1>Wahnsinn!</h1>
  <h2>Wie ich mit einer dämlichen Idee ein Vermögen machte</h2>
  <p>[&hellip;]</p>
  <p>am 13.08.2015 von Dr. B. Lödmann</p>`,
  `<h1>Prokrastination?</h1>
```

```
11   <h2>Wie oft hörst du dich selbst sagen: „Nein, ich hab's noch
nicht erledigt; ich habe es vor!"</h2>
12   <p>[&hellip;]</p>
13   <p>am 02.06.2015 von A. Meisenbär</p>`
14 ];
```

Codebeispiel 5.7 *Ausschnitt aus additional_files/05/examples/newsboard.js*

Hier ist noch das vorgegebene HTML:

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="utf-8" />
6
7    <title>Newsboard</title>
8
9    <meta name="viewport" content="width=device-width,
initial-scale=1.0" />
10   <link rel="stylesheet" type="text/css" href="newsboard.css"
media="screen" />
11   <script src="newsboard.js" defer="defer"></script>
12 </head>
13
14 <body>
15
16   <div class="newsboard_wrapper">
17     <div class="newsboard">
18       <span class="message_number">3</span>
19
20       <a href="#" class="close_button" title="Delete
message">&times;</a>
21
22       <div class="newsboard_content"></div>
23
24       <div class="paging_bar">
25         <span class="float_left">
26           <a href="#" title="first">«</a>
27           <a href="http://example.com" title="prev">‹</a>
28         </span>
29
30         <span class="float_right">
31           <a href="http://example.com" title="next">›</a>
32           <a href="#" title="last">»</a>
33         </span>
34
35         <span class="progressbar">
36           <progress max="3" value="1"
id="messages_progress"></progress>
37         </span>
38       </div>
39     </div>
40   </div>
41
42 </body>
43
44 </html>
```

Codebeispiel 5.8 *additional_files/05/examples/newsboard.html*

Die Nachrichten sollen im `div`-Element mit der Klasse `newsboard_content` erscheinen. Direkt nach dem Laden sollte die erste Nachricht bereits angezeigt werden. Der Code ist zunächst nichts Besonderes:

```
1 "use strict";
2 {
3   const init = () => {
4     showFirstMessage();
5   };
6   const showFirstMessage = () => showMessageByNumber(1);
7   const showMessageByNumber = messageNumber =>
8     $(".newsboard_content").innerHTML = messages[messageNumber -
1];
9   const messages = […];
10  const $ = document.querySelector.bind(document);
11  const $$ = document.querySelectorAll.bind(document);
12  init();
13 }
```

Bei der Initialisierung rufen Sie die Funktion `showFirstMessage` auf, die mittels `showMessageByNumber(1)` die erste Nachricht anzeigt. Letztere Funktion leistet die eigentliche Arbeit: Sie extrahiert die Nachricht aus dem Array und überträgt sie mittels `innerHTML` an die richtige Stelle im Dokument.

Einen kleinen Kniff zeigt Zeile 11. Als Index für das `messages`-Array verwenden Sie den Parameter `messageNumber`, ziehen aber vorher noch 1 ab. Das ist notwendig, weil das Array bei Index `0` beginnt. Wir möchten aber von der ersten Nachricht reden, nicht von der nullten.

Jetzt wird es ein wenig interessanter. Binden Sie das `click`-Event an die Vor- und Zurück-Zeichen `‹` und `›`. Sie können die Elemente, die die Zeichen enthalten, über das jeweilige `title`-Attribut `prev` bzw. `next` ansprechen.

```
1 …
2 const init = () => {
3   showFirstMessage();
4   $('[title=next]').addEventListener('click', nextMessage);
5   $('[title=prev]').addEventListener('click', prevMessage);
6 };
7 …
8 const nextMessage = e => console.log('next');
9 const prevMessage = e => console.log('prev');
10 …
```

Leider funktioniert das noch nicht. Der Designer hat nämlich Links (`a`-Elemente) für die Vor- und Zurück-Zeichen verwendet. Wären es `button`-Elemente gewesen, hätten Sie kein Problem. Aber Links haben bereits ein Verhalten beim Anklicken: Sie rufen die im `href`-Attribut angegebene URL auf!

Was tun? Sie könnten das Element natürlich ändern. Möglicherweise hat sich der Designer aber etwas dabei gedacht. Beispielsweise könnten die Links dafür da sein, dass das Newsboard auch ohne JS funktioniert — auf die klassische Weise.

Es gibt eine bessere Lösung. Verhindern Sie einfach, dass die Links ihr Standardverhalten (Default) auslösen.

Dafür stellt das Event-Objekt die Methode `preventDefault` zur Verfügung.

```
const nextMessage = e => {
  console.log('next');
  e.preventDefault();
};
const prevMessage = e => {
  console.log('prev');
  e.preventDefault();
};
```

Nachdem dieses Problem gelöst ist, können Sie sich darum kümmern, tatsächlich die Nachrichten umzublättern, statt nur Log-Nachrichten auszugeben!

Die Kernproblematik ist hier, dass Sie sich die Nummer der gerade angezeigten Nachricht irgendwo merken müssen. Das heißt, die Anwendung hat einen ***Zustand*** (engl. ***state***). Das Drücken des Vorwärtszeichens ist abhängig von diesem Zustand. Das Programm muss wissen, welche Nachricht gerade angezeigt wird, um die nächste zu bestimmen.

Zustandsbehaftung ist immer etwas problematisch. Sie erschwert das Programmverständnis. Im Grunde wollen wir einen veränderlichen Zustand (***mutable State***) bei der Programmierung vermeiden. Gerade in grafischen Oberflächen ist er aber oft ein notwendiges Übel.

Im Moment haben Sie noch keine gute Lösung, um damit umzugehen — wir kommen später nochmal darauf zurück. Deswegen verwenden Sie erstmal eine Lösung, die nicht ideal ist. Sie können später immer noch ein Refactoring durchführen.

Speichern Sie die Nummer der aktuellen Nachricht in einer Variable, die sich mehrere Funktionen teilen:

```
let currentMessageNumber = 1;
```

Damit haben Sie eine einfache Möglichkeit, den Wert beim Vor- und Zurückblättern zu ändern.

```
{
  ...
  const nextMessage = e => {
    showMessageByNumber(currentMessageNumber += 1);
    e.preventDefault();
  };
  const prevMessage = e => {
    showMessageByNumber(currentMessageNumber -= 1);
    e.preventDefault();
  };
```

```
11    ...
12    let currentMessageNumber = 1;
13    ...
14  }
```

Die Variable `currentMessageNumber` hat zum Glück keinen ***global-Scope***, da sie nur innerhalb des äußeren Blocks existiert. Solange dieser Block nicht sehr viel größer wird, ist das tolerierbar. Sobald Ihre Anwendung wächst, brauchen Sie aber eine bessere Lösung. Wir kümmern uns in Lektion 7 darum.

Hurra, es funktioniert! Hier ist der Code nochmal im Ganzen:

```
1  "use strict";
2
3  {
4    const init = () => {
5      showFirstMessage();
6      $("[title=next]").addEventListener("click", nextMessage);
7      $("[title=prev]").addEventListener("click", prevMessage);
8    };
9
10   const showFirstMessage = () => showMessageByNumber(1);
11
12   const nextMessage = e => {
13     showMessageByNumber(currentMessageNumber += 1);
14     e.preventDefault();
15   };
16   const prevMessage = e => {
17     showMessageByNumber(currentMessageNumber -= 1);
18     e.preventDefault();
19   };
20
21   const showMessageByNumber = messageNumber =>
22     $(".newsboard_content").innerHTML = messages[messageNumber -
1];
23
24   const $ = document.querySelector.bind(document);
25
26   const messages = [
27     `<h1>Tutoren streiken!!!</h1>
28     <h2>Alle Einsendeaufgaben werden ab sofort mit 0 Punkten
bewertet</h2>
29     <p>Überall dieselbe alte Leier. Das Layout ist fertig, der
Text lässt auf sich warten. Damit das Layout nun nicht nackt im Raume
steht und sich klein und leer vorkommt, springe ich ein: der
Blindtext. Genau zu diesem Zwecke erschaffen, immer im Schatten
meines großen Bruders »Lorem Ipsum«, freue ich mich jedes Mal, wenn
Sie ein paar Zeilen lesen.</p>
30     <p class="newsboard_footer">am 25.09.2015 von K. Einer</p>`,
31
32     `<h1>Wahnsinn!</h1>
33     <h2>Wie ich mit einer dämlichen Idee ein Vermögen machte</h2>
34     <p>Polyfon zwitschernd aßen Mäxchens Vögel Rüben, Joghurt und
Quark. "Fix, Schwyz! " quäkt Jürgen blöd vom Paß. Victor jagt zwölf
Boxkämpfer quer über den großen Sylter Deich. Falsches Üben von
Xylophonmusik quält jeden größeren Zwerg. Heizölrückstoßabdämpfung.
Zwei flinke Boxer jagen die quirlige Eva und ihren Mops durch
```

```
Sylt.</p>
35      <p class="newsboard_footer">am 13.08.2015 von Dr. B.
Lödmann</p>`,
36
37      `<h1>Prokrastination?</h1>
38      <h2>Wie oft hörst du dich selbst sagen: „Nein, ich hab's noch
nicht erledigt; ich habe es vor!"</h2>
39      <p>Denn esse est percipi - Sein ist wahrgenommen werden. Und
weil Sie nun schon die Güte haben, mich ein paar weitere Sätze lang
zu begleiten, möchte ich diese Gelegenheit nutzen, Ihnen nicht nur
als Lückenfüller zu dienen, sondern auf etwas hinzuweisen, das es
ebenso verdient wahrgenommen zu werden: Webstandards nämlich.</p>
40      <p>Sehen Sie, Webstandards sind das Regelwerk, auf dem
Webseiten aufbauen. So gibt es Regeln für HTML, CSS, JavaScript oder
auch XML; Worte, die Sie vielleicht schon einmal von Ihrem Entwickler
gehört haben. Diese Standards sorgen dafür, dass alle Beteiligten aus
einer Webseite den größten Nutzen ziehen. Im Gegensatz zu früheren
Webseiten müssen wir zum Beispiel nicht mehr zwei verschiedene
Webseiten für den Internet Explorer und einen anderen Browser
programmieren.</p>
41      <p class="newsboard_footer">am 02.06.2015 von A. Meisenbär</p>`
42    ];
43
44    let currentMessageNumber = 1;
45
46    init();
47  }
```

Codebeispiel 5.9 *additional_files/05/examples/newsboard.js*

Ein paar Dinge lassen sich aber verbessern. Darum können Sie sich gleich in den Übungen kümmern.

5.7 Kurz und knapp — on

Genau wie `document.querySelectorAll` ist auch `.addEventHandler` oft umständlich. Viele Bibliotheken (z.B. jQuery) haben dafür die Methode `on` geprägt. So bedeutet beispielsweise `$("#my_button").on('click', ...)` soviel wie »beim Klicken (*on click*) auf den Button mit der id *my_button*«. Da wir natürlich auch in den Genuss einer vereinfachten Schreibweise kommen möchten, verwenden wir wieder einmal Code, der ursprünglich aus *bling.js*[29] stammt:

```
Node.prototype.on = function (name, fn) {
  this.addEventListener(name, fn);
  return this;
};

NodeList.prototype.on = NodeList.prototype.addEventListener =
```

29. *https://gist.github.com/paulirish/12fb951a8b893a454b32*

```
function (name, fn) {
  this.forEach(elem  => elem.on(name, fn));
  return this;
};
```

Damit ist `on` sowohl auf `Node` als auch auf `NodeList` verfügbar und damit auf allem, was Sie mittels $ und $$ selektieren können. Hier ist ein Beispiel für die Verwendung auf `NodeList`:

Für einen *Online-Taschenrechner* soll ein Tastenfeld erstellt werden, bei dem die jeweils gedrückte Zahl im Display erscheint. Hier das HTML:

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="UTF-8" />
6    <title>Calculator</title>
7
8    <script src="calculator.js" defer="defer"></script>
9
10   <style type="text/css">
11     #display {
12       width: 140px;
13     }
14
15     #keyfield {
16       width: 130px;
17       padding: 5px;
18       border: 1px solid black;
19     }
20
21     #keyfield > button {
22       width: 40px;
23       height: 40px;
24
25     }
26   </style>
27 </head>
28
29 <body>
30   <p><input type="text" id="display"/></p>
31   <div id="keyfield">
32     <button>7</button>
33     <button>8</button>
34     <button>9</button>
35     <button>4</button>
36     <button>5</button>
37     <button>6</button>
38     <button>1</button>
39     <button>2</button>
40     <button>3</button>
```

```
41   </div>
42 </body>
43
44 </html>
```

Codebeispiel 5.10 *additional_files/05/examples/calculator/calculator.html*

Im JS benötigen Sie (neben unseren Hilfsfunktionen) jetzt nur noch eine einzige Zeile Code, um die gewünschte Funktionalität zu realisieren:

```
$$("#keyfield > button").on("click",
  e => $("#display").value += e.target.innerHTML);
```

Die übergebene Funktion

```
e => $("#display").value += e.target.innerHTML
```

schreibt die jeweilige Zahl ins `display`. Statt diese Funktion auf jeden Button einzeln zu registrieren, genügt ein einziges `on`, um alle Buttons mit dem Event-Handler auszustatten.

Wenn Sie möchten, können Sie sogar mehrere `.on`-Aufrufe verketten:

```
$$("#keyfield > button")
  .on("click", e => $("#display").value += e.target.innerHTML)
  .on("mousenter", e => console.log(e.target.innerHTML));
```

5.8 Zusammenfassung

Eventname	wird ausgelöst, wenn …	z. B. verfügbar auf
blur	ein Element den Fokus verliert.	input, textarea
change	ein Anwender eine Änderung am Element bewirkt — z. B. durch Wählen einer Option innerhalb einer `select`-Box oder durch das Bestätigen einer Checkbox. Im Gegensatz zum `input`-Event gilt: Bei einem `input`-Field oder einer `textarea` »feuert« dieses Event erst, nachdem das Feld seinen Fokus wieder verliert (d. h. die Eingabe als abgeschlossen gilt).	input, select, textarea
click	ein Anwender auf ein Element klickt (und wieder loslässt). Das `click`-Event repräsentiert den kompletten Vorgang: Drücken und Loslassen. Wenn Sie nur auf das Runterdrücken reagieren möchten, verwenden Sie `mousedown`, für nur das Loslassen `mouseup`.	button, img, body, p, div
dblclick	ein Anwender auf einem Element einen Doppelklick ausführt.	button, img, body
focus	ein Element den Fokus erhält.	input, textarea
hashchange	sich der Fragmentbezeichner der URL ändert (der Fragmentbezeichner ist der Teil, der nach dem #-Symbol folgt, inkl. Symbol).	window

Tabelle 5.2 *Ein Auswahl in der Praxis benötigter Browser-Events. Eine vollständige Liste finden Sie in der MDN Event reference*[30].

30. *https://developer.mozilla.org/en-US/docs/Web/Events*

Eventname	wird ausgelöst, wenn ...	z. B. verfügbar auf
input	der Inhalt eines `input`- oder `textarea`-Elements geändert wird. Im Gegensatz zu `change` feuert dieser Event bei jeder Eingabe sofort. Unter *jsfiddle.net/AtvtZ/* finden Sie einen guten Vergleich von `input` und `change`.	input, textarea
keydown	der Anwender eine Taste drückt.	input, textarea
keypress	der Anwender eine Taste gedrückt hält.	input, textarea
keyup	der Anwender eine Taste loslässt.	input, textarea
mouseenter	der Anwender den Mauszeiger über das Element bewegt — genauer: die Fläche des Elements mit dem Mauszeiger betritt.	button, img, body, p, div
mouseleave	der Anwender den Mauszeiger aus einem Element herausbewegt.	button, img, body, p, div
mousemove	sich der Mauszeiger über dem Element bewegt.	button, img, body, p, div
reset	ein Formular zurückgesetzt wird.	form
resize	der Anwender die Größe des Browserfensters verändert.	window
scroll	innerhalb eines Elements (oder des Browserfensters) gescrollt wird.	window, div
select	der Anwender Text markiert.	input, textarea, p
submit	ein Formular abgeschickt wird.	form

Tabelle 5.2 *Ein Auswahl in der Praxis benötigter Browser-Events. Eine vollständige Liste finden Sie in der MDN Event reference.*

Eventname	wird ausgelöst, wenn …	z. B. verfügbar auf
transitionend	eine CSS-Transition beendet ist.	img, div, p
visibilitychange	Inhalt eines Elements sichtbar oder unsichtbar wird.	img, div, p
wheel	der Anwender am Rad dreht (wir meinen das wörtlich, nicht im übertragenen Sinn). Normalerweise ist es das Mausrad, aber es kann auch ein Trackball oder ein Touchpad sein.	window, div

Tabelle 5.2 *Ein Auswahl in der Praxis benötigter Browser-Events. Eine vollständige Liste finden Sie in der MDN Event reference.*

Attribut	Bedeutung
altKey	Gibt `true` bei gedrückter **ALT**-Taste zurück.
button	Nummer des gedrückten Buttons (0 bis 4)
clientX	X-Position des Mauszeigers im DOM
clientY	Y-Position des Mauszeigers im DOM
ctrlKey	Gibt `true` bei gedrückter **CTRL**-Taste zurück.
metaKey	Gibt `true` bei gedrückter **META**-Taste (Windows- bzw. Command-Taste) zurück.
movementX	Relative Bewegung des Mauszeigers auf der X-Coordinate seit dem letzten `mousemove`
movementY	Relative Bewegung des Mauszeigers auf der Y-Coordinate seit dem letzten `mousemove`
screenX	X-Position des Mauszeigers auf dem Bildschirm
screenY	Y-Position des Mauszeigers auf dem Bildschirm
shiftKey	Gibt `true` bei gedrückter **SHIFT**-Taste zurück.

Tabelle 5.3 MouseEvent*: wichtige Attribute*

Attribut	Bedeutung
altKey	Gibt `true` bei gedrückter **ALT**-Taste zurück.
code	Codewert der gedrückten Taste, z. B. `KeyA`
ctrlKey	Gibt `true` bei gedrückter **CTRL**-Taste zurück.
key	Codewert der gedrückten Taste, z. B. `a`
metaKey	Gibt `true` bei gedrückter **META**-Taste (Windows- bzw. Command-Taste) zurück.
shiftKey	Gibt `true` bei gedrückter **SHIFT**-Taste zurück.

***Tabelle 5.4 KeyboardEvent**: wichtige Attribute*

Attribut	Bedeutung
deltaX	Scrollbewegung auf der X-Achse
deltaY	Scrollbewegung auf der Y-Achse

***Tabelle 5.5 WheelEvent**: wichtige Attribute*

Attribut	Bedeutung
currentTarget	Element, auf dem der aktuelle Eventhandler registriert ist (oft identisch mit `target`)
target	Element, das das Event ausgelöst hat (oft identisch mit `currentTarget`)
timestamp	Zeitpunkt, zu dem das Event ausgelöst wurde

***Tabelle 5.6 Event**: wichtige Attribute (werden an alle speziellen Eventtypen, wie z. B. `MouseEvent` vererbt)*

5.9 Übungen

Übung 10: Newsboard: Das geht noch besser

Das Newsboard wartet noch auf das Vervollständigen seiner Funktionen.

1. Belegen Sie die Buttons `»` und `«` mit Funktionalität. Der Button `«` soll es erlauben, zur ersten Nachricht zu springen, der Button `»` zur letzen.

2. Der bisher eher sinnlose rote Kreis in der linken oberen Ecke soll nun die Gesamtanzahl der Nachrichten anzeigen. Schreiben Sie eine Funktion, die diese Anzahl ausliest und in das `<span>` mit der Klasse `message_number` einträgt.

3. Unterstützen Sie ab sofort das Navigieren durch die News mit der Tastatur. Die Pfeiltasten nach links und rechts sollen dabei jeweils die gleiche Funktionalität wie die Buttons `<` und `>` ermöglichen.

4. Auch das Springen auf die erste bzw. letzte Position soll nun mit Tastatur möglich sein. Die Tastenkombination dafür ist **crtl** und die Pfeiltaste nach links (zur ersten Nachricht) bzw. rechts (zur letzten Nachricht).

Mehrere JS-Dateien verwenden

6

Hi. Ich komme gerade von einem Meeting mit Carl Biersee. Carl ist unser Consultant. Wir zahlen ihm eine Menge Geld dafür, dass er uns immer wieder sagt, wie toll unsere neue Plattform ist, und dass wir auf dem richtigen Weg sind.

Heute hat er einen Blick auf die aktuelle Codebasis geworfen. Ich meine, Carl ist ein netter Kerl. Er wollte den Code nicht schlecht reden. Aber er hat uns doch klar zu verstehen gegeben, dass da nach oben noch ziemlich viel Luft ist.

Ein Sache, die Sie auf jeden Fall verbessern sollten: den Code auf mehrere Dateien aufteilen. Er hat auch noch irgendwas von DRY und Rädern erzählt ... oder so ähnlich?

Ob Karl sein Geld wirklich wert ist, darüber kann man sicherlich geteilter Meinung sein. In einem hat er jedoch recht: Ab einer gewissen Größe ist es sinnvoll, den Code auf mehrere Dateien aufzuteilen. Das bringt eine Reihe von Vorteilen:

- bessere Übersicht
- einfachere Wiederverwendung
- erleichterte Teamarbeit

Bessere Übersicht

Je größer eine Datei wird, um so schwieriger ist es, innerhalb der Datei die richtige Codestelle zu finden. Wenn Sie Ihren Code thematisch aufteilen und die Dateien sinnvoll benennen, hilft das ungemein bei der Wartung und Weiterentwicklung.

Stellen Sie sich vor, Sie sind Handwerker und suchen ein Werkzeug. In einem möglichen Szenario haben Sie einen Schreibtisch mit beschrifteten Schubladen. In

einem anderen haben Sie eine große Kiste, in die das Werkzeug reingekippt wurde. Wo würden Sie lieber suchen?

Einfachere Wiederverwendung

Wenn Sie Funktionen haben, die Sie an vielen Stellen in Ihrer Anwendung benötigen, ist es natürlich ungeschickt, das Rad immer wieder aufs Neue zu erfinden. Auch *Copy&Paste* von Funktionen ist keine besonders gute Lösung. Wir haben das zwar in der Vergangenheit immer wieder getan. Aber jetzt wollen wir das verbessern.

Das Problem mit Copy&Paste zeigt sich spätestens dann, wenn Sie Ihren Code überarbeiten oder einen Bug ausmerzen. In dem Fall müssen Sie alle Kopien dieses Codefragments ebenfalls anpassen. Tun Sie das nicht, wird der Bug Sie verfolgen, oder es kommt zu Inkonsistenzen und überraschendem Verhalten.

Besser ist: Sie lagern gemeinsamen Code aus und binden die gleiche Datei an allen benötigten Stellen ein. Dann vermeiden Sie Inkonsistenzen und müssen nicht den Klonen hinterjagen.

Das DRY-Prinzip

DRY ist hier nicht wörtlich gemeint. Es steht für ***D**on't **R**epeat **Y**ourself* — wiederhole Dich nicht! Damit ist gemeint, dass Entwickler immer versuchen sollten, Redundanzen im Code zu vermeiden.

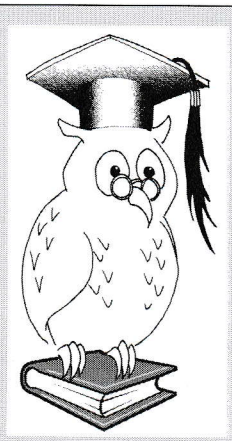
Es geht nicht um simple Duplikate. Jedes Stück Wissen, jedes Konzept sollte genau **eine** autoritative Quelle, nur **eine** Repräsentation im Code haben (Venners 2003). Wenn sich ein Konzept ändert, sollte es im Code idealerweise nur **eine** Stelle geben, die Sie anpassen müssen. Manchmal ist es strittig, ob Code redundant ist oder nur zufällig gleich.

Mittels Copy&Paste erzeugte Funktionen verstoßen meistens gegen das Dry-Prinzip. **Vermeiden Sie Copy&Paste!**

Erleichterte Teamarbeit

Falls Sie mit mehreren Leuten am gleichen Projekt arbeiten, ist es hilfreich, viele kleine statt wenige große Dateien zu haben. Je besser das Projekt aufgeteilt ist, desto weniger kommen Sie sich in die Quere.

Gute Versionierungssysteme (wie z. B. *GIT*[31]) helfen, Änderungen zusammenzuführen, wenn mehrere Entwickler an der selben Datei arbeiten. Am einfachsten ist es aber immer noch, wenn das gar nicht nötig ist.



Die empfohlenen Dateigröße liegt übrigens bei ca. **50 bis 60 Zeilen Code**. Das ist etwa die Codemenge, die Sie ohne zu scrollen auf eine Bildschirmseite bekommen. Das ist natürlich nur eine Daumenregel — begründete Ausnahmen gibt es immer wieder.

6.1 Fröhliches Datei-Zerhacken

Wie lässt sich das nun realisieren? Im Grunde ist es nicht schwierig, aber es gibt ein paar Stolperfallen. Oft ist die Reihenfolge, in der das HTML-Dokument die JS-Dateien einbindet, relevant.

Das `defer`-Attribut sorgt dafür, dass das Laden der Scripte asynchron geschieht und die Scripte somit in (mehr oder weniger) zufälliger Reihenfolge eintrudeln. Glücklicherweise garantiert das `defer`-Attribut aber auch, dass die Scripte nach dem Laden in der Reihenfolge ausgeführt werden, die*das HTML-Dokument vorgibt*[32]. Damit unterscheidet es sich vom `async`-Attribut, das Scripte sofort nach dem Laden ausführt — völlig unabhängig von der Reihenfolge der `script`-Tags im HTML.

```
1 <script src="code1.js" defer="defer"></script>
2 <script src="code2.js" defer="defer"></script>
3 <script src="code3.js" defer="defer"></script>
```

Dennoch ist es aus Wartungsgründen unschön, wenn eine Änderung der Reihenfolge dazu führt, dass das Programm nicht mehr funktioniert. Mit ein wenig Disziplin können Sie das vermeiden.

31. *https://git-scm.com*
32. *http://www.breck-mckye.com/blog/2014/04/document-loading-and-DOM-lifecycle-events/*

Halten Sie einfach folgende Regeln ein:

- Aufrufe von Funktionen erfolgen nur in der letzten eingebundenen Datei. Idealerweise gibt es nur einen einzigen Funktionsaufruf, z. B. `init()` oder `run()`.
- Alle anderen Dateien definieren lediglich Variablen (`let`) oder — bevorzugt — Konstanten (`const`).

Wenn Sie die beiden obigen Regeln befolgen, sind Sie bereits auf der sicheren Seite. Die Funktionalität ist komplett geladen, bevor ein Funktionsaufruf sie benötigt. Darüber hinaus gibt es Programmierrichtlinien, die Ihnen helfen, den Code sinnvoll zu organisieren.

Programmierrichlinie

- Ordnen Sie JS-Dateien von *generisch* nach *spezifisch*.

Die letzte Datei, die Sie einbinden, sollte den speziellsten Code enthalten, der sich auf die aktuelle Anwendung bezieht. Die davor eingebundenen Dateien können allgemeinere Funktionen enthalten, die auch an anderer Stelle benötigt werden.

- Laden Sie externe Bibliotheken und Fremdcode immer vor dem eigenen Code.

Falls Sie mehrere unabhängige JS-Funktionalitäten in die gleiche HTML-Seite einbinden, so betrachten Sie jede dieser Funktionalitäten als eine eigenständige Gruppe von Scripten. Die obigen Regeln gelten dann pro Gruppe.

Beispiel

```
1 <script src="gruppe1_allgemeine_bibliothek1.js"
defer="defer"></script>
2 <script src="gruppe1_allgemeine_bibliothek2.js"
defer="defer"></script>
3 <script src="gruppe1_eigene_funktionen.js" defer="defer"></script>
4 <script src="gruppe1_eigene_datei_mit_init_aufruf.js"
defer="defer"></script>
5 <script src="gruppe2_allgemeine_bibliothek1.js"
defer="defer"></script>
6 <script src="gruppe2_allgemeine_bibliothek2.js"
defer="defer"></script>
7 <script src="gruppe2_eigene_funktionen.js" defer="defer"></script>
8 <script src="gruppe2_eigene_datei_mit_init_aufruf.js"
defer="defer"></script>
```

Damit Carl nun endlich Ruhe gibt (Sie wissen schon, er wird dafür bezahlt, dass er »Probleme« findet), teilen Sie das Newsboard am besten in drei Dateien auf.

```
1 "use strict";
2
3 const $ = document.querySelector.bind(document);
```

Codebeispiel 6.1 *additional_files/06/examples/newsboard/dom_helper.js*

Die Datei *dom_helper.js* enthält im Moment nur die $-Funktion. Wenn Sie weitere allgemeine Funktionen benötigen, können Sie sie hier ergänzen.

```
1 "use strict";
2
3 const messages = [
4   `<h1>Tutoren streiken!!!</h1>
5   <h2>Alle Einsendeaufgaben werden ab sofort mit 0 Punkten
bewertet</h2>
6   <p>Überall dieselbe alte Leier. Das Layout ist fertig, der Text
lässt auf sich warten. Damit das Layout nun nicht nackt im Raume
steht und sich klein und leer vorkommt, springe ich ein: der
Blindtext. Genau zu diesem Zwecke erschaffen, immer im Schatten
meines großen Bruders »Lorem Ipsum«, freue ich mich jedes Mal, wenn
Sie ein paar Zeilen lesen.</p>
7   <p class="newsboard_footer">am 25.09.2015 von K. Einer</p>`,
8
9   `<h1>Wahnsinn!</h1>
10  <h2>Wie ich mit einer dämlichen Idee ein Vermögen machte</h2>
11  <p>Polyfon zwitschernd aßen Mäxchens Vögel Rüben, Joghurt und
Quark. "Fix, Schwyz! " quäkt Jürgen blöd vom Paß. Victor jagt zwölf
Boxkämpfer quer über den großen Sylter Deich. Falsches Üben von
Xylophonmusik quält jeden größeren Zwerg. Heizölrückstoßabdämpfung.
Zwei flinke Boxer jagen die quirlige Eva und ihren Mops durch
Sylt.</p>
12  <p class="newsboard_footer">am 13.08.2015 von Dr. B.
Lödmann</p>`,
13
14  `<h1>Prokrastination?</h1>
15  <h2>Wie oft hörst du dich selbst sagen: „Nein, ich hab's noch
nicht erledigt; ich habe es vor!"</h2>
16  <p>Denn esse est percipi - Sein ist wahrgenommen werden. Und
weil Sie nun schon die Güte haben, mich ein paar weitere Sätze lang
zu begleiten, möchte ich diese Gelegenheit nutzen, Ihnen nicht nur
als Lückenfüller zu dienen, sondern auf etwas hinzuweisen, das es
ebenso verdient wahrgenommen zu werden: Webstandards nämlich.</p>
17  <p>Sehen Sie, Webstandards sind das Regelwerk, auf dem Webseiten
aufbauen. So gibt es Regeln für HTML, CSS, JavaScript oder auch XML;
Worte, die Sie vielleicht schon einmal von Ihrem Entwickler gehört
haben. Diese Standards sorgen dafür, dass alle Beteiligten aus einer
Webseite den größten Nutzen ziehen. Im Gegensatz zu früheren
Webseiten müssen wir zum Beispiel nicht mehr zwei verschiedene
Webseiten für den Internet Explorer und einen anderen Browser
programmieren.</p>
18  <p class="newsboard_footer">am 02.06.2015 von A. Meisenbär</p>`
19 ];
```

Codebeispiel 6.2 *additional_files/06/examples/newsboard/messages.js*

Die Datei *message.js* enthält die Nachrichten selbst. Falls jemand weitere Nachrichten ergänzt, muss er nur noch diese Datei ändern. Das vermindert die Wahrscheinlichkeit, dass sich nachträglich Fehler in den eigentlichen Anwendungscode einschleichen.

```
1  "use strict";
2
3  {
4    const init = () => {
5      showFirstMessage();
6      $("[title=next]").addEventListener("click", nextMessage);
7      $("[title=prev]").addEventListener("click", prevMessage);
8    };
9
10   const showFirstMessage = () => showMessageByNumber(1);
11
12   const nextMessage = e => {
13     showMessageByNumber(currentMessageNumber += 1);
14     e.preventDefault();
15   };
16   const prevMessage = e => {
17     showMessageByNumber(currentMessageNumber -= 1);
18     e.preventDefault();
19   };
20
21   const showMessageByNumber = messageNumber =>
22     $(".newsboard_content").innerHTML = messages[messageNumber -
1];
23
24   let currentMessageNumber = 1;
25
26   init();
27 }
```

Codebeispiel 6.3 *additional_files/06/examples/newsboard/newsboard.js*

Folgender Code bindet die drei Dateien ein:

```
1 <head>
2   ...
3   <script src="dom_helper.js" defer="defer"></script>
4   <script src="messages.js" defer="defer"></script>
5   <script src="newsboard.js" defer="defer"></script>
6   ...
7 </head>
8 ...
```

Codebeispiel 6.4 *newsboard.html*

6.2 Der Terror der globalen Verschmutzung

Leider gibt es ein weiteres Hindernis. Wenn Sie auf Variablen und Konstanten aus einer anderen Datei zugreifen möchten, dürfen Sie diese nicht in einen Block packen. Ein äußerer Block sorgt ja gerade dafür, dass die Namen der Variablen

(bzw. Konstanten) diesen nicht verlassen und damit den Sichtbarkeitsbereich der Datei überschreiten.

Das kann aber unweigerlich zu den Problemen führen, vor denen gerade der Block schützen sollte: der *Verschmutzung des globalen Namensraums*.

Binden Sie viele Dateien ein, kann es passieren, dass diese die gleichen Namen verwenden. In dem Fall kommt es zu einem Namenskonflikt und dadurch zu Fehlern. Vor allem, wenn Sie viele 3rd-Party-Scripts (z.B. Youtube, Google-Analytics) oder externe Bibliotheken einbinden, kann es schnell passieren, dass Sie die Kontrolle verlieren. Dann steht Ihnen eine aufwendige Fehlersuche bevor.

Das klingt möglicherweise erst einmal nach einer eher unwahrscheinlichen Situation. Wir können Ihnen aber versichern, dass das ein sehr reales Problem ist. Es ist nicht so selten, wie Sie vielleicht denken!

Versuchen Sie deswegen die Anzahl der globalen Variablen und Konstanten möglichst klein zu halten.

Ein kleines Experiment

Suchen Sie sich einen beliebigen erfahrenen JS-Entwickler in Ihrer Nachbarschaft (falls nicht vorhanden, Radius bitte ausdehnen). Bitten Sie diesen Entwickler, eine Anekdote zu Thema *Namenskonflikt* zum Besten zu geben. Vermutlich wird er Ihnen eine haarsträubende Geschichte erzählen.

Kampf gegen die globale Verschmutzung

Tatsächlich gibt es Auswege aus dem Dilemma der *globalen Namensraumverschutzung*. Die entsprechenden Prinzipien sprengen aber ein wenig den Rahmen dieses Buches. Falls es Sie interessiert, finden Sie hier ein paar weiterführende Infos:

- *Modulpattern & revealing Modulepattern*[33] (Osmani)
- *ES6-Imports mit webpack und Babel*[34] (Rauschmayer 2016)
- *ES6-Imports mit dem ES6-module-loader*[35]

Übung 11:

Zerlegen Sie den Code aus Codebeispiel 5.5 nach den Prinzipien dieser Lektion.

6.3 Kein bisschen verstaubte Bibliotheken

Im Zusammenhang mit Programmiersprachen ist eine ***Bibliothek*** (engl. ***library*** — abgekürzt ***lib***) eine Sammlung von Code (z.B. in Form von Funktionen), den Sie in Ihre eigenen Programme einbinden können. Der Code kann dabei unterschiedlich spezialisiert oder allgemein gehalten sein. Normalerweise ist er aber eher generisch und kann in vielen verschiedenen Programmen verwendet werden.

Im Grunde haben Sie in dieser Lektion schon eine Bibliothek zusammengestellt — vielleicht nicht aus dicken Wälzern und alten Scharteken, aber in Form einer Datei. Genau! Die Datei *dom_helper.js* ist bereits eine kleine Bibliothek.

```
1 "use strict";
2
3 const $ = document.querySelector.bind(document);
```

Codebeispiel 6.5 *additional_files/06/examples/newsboard/dom_helper.js*

Zugegeben, sie enthält momentan nur eine Funktion — die Funktion `$`. Eine Bibliothek mit nur einem Buch würden Sie wohl kaum eine Bibliothek nennen. Für nur eine einzige Funktion wäre es auch in der Praxis ein wenig übertrieben, eine Bibliothek einzubinden. Tatsächlich haben wir aber bereits eine Reihe von Funktionen und Erweiterungen, die wir häufiger benötigen.

Tragen Sie diese Funktionen in die Datei *dom_helper.js* ein, und Sie können in Zukunft auf ständiges Copy&Paste dieser Funktionen verzichten.

```
1 "use strict";
2
3 const $ = document.querySelector.bind(document);
4 const $$ = document.querySelectorAll.bind(document);
5
6 NodeList.prototype.__proto__ = Array.prototype;
7 HTMLCollection.prototype.__proto__ = Array.prototype;
8
9 Node.prototype.on = function(name, fn) {
```

33. *https://addyosmani.com/resources/essentialjsdesignpatterns/book/#modulepatternjavascript*
34. *https://leanpub.com/setting-up-es6/read#sec_webpack-babel*
35. *https://github.com/ModuleLoader/es6-module-loader*

```
10    this.addEventListener(name, fn);
11    return this;
12  };
13
14  NodeList.prototype.on = NodeList.prototype.addEventListener =
function(name, fn) {
15    this.forEach(elem => elem.on(name, fn));
16    return this;
17  };
```

Codebeispiel 6.6 *additional_files/lib/dom_helper.js*

Im Begleitmaterial finden Sie die Datei bereits im `/lib`-Verzeichnis.

Falls Sie einen Fehler finden, müssen Sie ihn auch nur einmal beheben und nicht in allen durch Copy&Paste entstandenen Kopien.

Sharing is caring: Einbinden fremder Bibliotheken

Das Gute ist, dass bereits viele JS-Entwickler Bibliotheken zu den verschiedensten Themen angelegt und unter einer OpenSource-Lizenz veröffentlicht haben. Solche Bibliotheken können Sie bei Bedarf einbinden und müssen dadurch nicht alles selbst entwickeln.

Package Manager

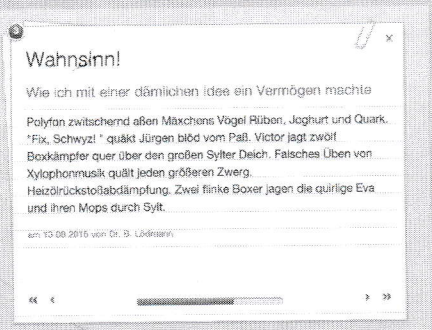
Das Einbinden und Wiederverwenden von fremden Bibliotheken gelingt am einfachsten mit Hilfe eines ***Package-Managers***. Das Verwenden eines solchen Managers ist aber kein Thema dieses Grundlagenbuchs. Falls Sie mehr darüber wissen möchten, finden Sie unter diesen URLs zwei bekannte Package-Manager für JS.

- *https://www.npmjs.com*
- *jspm.io*

7 Attribute: Erkennen, Ändern und Manpulieren ausdrücklich erlaubt

Wir hätten da gerne noch eine Verbesserung für den Newsbereich. Es ist wirklich nur eine Kleinigkeit.

Zur besseren Visualisierung haben wir uns vorgestellt, dass ein Fortschrittsbalken anzeigt, wie viele News der Anwender schon gelesen hat und wie viele noch kommen. Das können Sie doch bestimmt heute noch einbauen, oder?



Tja, mal sehen, ob Sie das heute fertig bekommen. Bevor Sie sich aber der Aufgabe im Ganzen widmen können, möchten wir Ihnen zeigen, wie Sie einen Fortschrittsbalken (engl. progress bar) mit JS ansprechen. Einen Fortschrittsbalken können Sie mit dem HTML5-Element `progress` erzeugen:

```
<progress value="3" max="10"></progress>
```

Codebeispiel 7.1 *das HTML-Element progress*

Das Attribut `max` gibt den maximalen Wert des Balkens an, bei dem er am Ende angelangt ist. Mit `value` ist der aktuelle Wert gemeint. Wenn Sie also gerade

Nachricht 3 von 10 anzeigen möchten, so müssen Sie `max` auf `10` und `value` auf `3` setzen, wie in Codebeispiel 7.1.

Natürlich benötigen Sie jetzt die Möglichkeit, den Wert des `value`-Attributs mit JavaScript zu ändern.

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="utf-8" />
6    <title>Progressbar</title>
7  </head>
8
9  <body>
10   <progress id="messages_progress" value="3" max="10"></progress>
11 </body>
12
13 </html>
```

Codebeispiel 7.2 *additional_files/07/examples/progressbar.html*

Öffnen Sie die Datei zu Codebeispiel 7.2 im Browser und greifen Sie auf das Element-Objekt des Balkens anhand seiner id `messages_progress` zu:

```
$('#messages_progress')
```

Alle HTML-Element-Objekte bieten die Methoden `getAttribute` und `setAttribute` zum Auslesen und Verändern von Attributwerten an.

Mittels

```
$('#messages_progress').getAttribute('value')
```

erhalten Sie den aktuellen Wert `3` des `value`-Attributs zurück und mittels

```
$('#messages_progress').setAttribute('value', 8)
```

können Sie den Wert auf `8` ändern.

Element-Objekte verfügen außerdem über die Eigenschaft `attributes`, die eine Liste aller verfügbaren Attribute zurückliefert:

```
$('#messages_progress').attributes // => NamedNodeMap [ max="20",
value="7", id="messages_progress" ]
```

Der Rückgabewert ist eine sogenannte `NamedNodeMap` — ein Objekt, das alle Attribute und deren Werte enthält.

7.1 IDL-Attribut vs. Content-Attribut

Generell können Sie mit den beiden Methoden `getAttribute` und `setAttribute` nahezu beliebige Attribute auf HTML-Elementen auslesen und ändern. In vielen Fällen stellt das DOM die Attribute aber auch direkt als JS-Properties auf dem HTML-Element-Objekt zur Verfügung. So können Sie z. B. den Wert des Balkens auch direkt mit

```
$('#messages_progress').value
```

auslesen bzw. mit

```
$('#messages_progress').value = 8
```

auf `8` ändern. Die Attribute, die Sie mit den Zugriffs-Methoden auslesen, heißen ***Content-Attribute*** und die, die Sie direkt über Eigenschaften ansprechen, werden ***IDL-Attribute*** genannt. In den meisten Fällen macht es keinen Unterschied, welche Art von Attribut Sie verwenden. Die IDL-Attribute greifen auf die zugrundeliegenden Content-Attribute zu.

Einen kleinen Unterschied stellt der verwendete Datentyp dar. Content-Attribute sind immer vom Datentyp `string`.

```
typeof $('#messages_progress').getAttribute('value') // => string
```

IDL-Attribute können dagegen unterschiedliche Datentypen haben. So ist z. B. der `value` des `progress`-Elementes vom Typ `number`.

```
typeof $('#messages_progress').value // => number
```

7.2 Es lebe der Fortschritt(sbalken)!

Jetzt, wo Sie wissen, wie es funktioniert, müssen Sie den Fortschrittsbalken nur noch anbinden. Das Schöne dabei ist, dass Sie die ungeliebte Variable `currentMessageNumber` dabei loswerden. Statt in der Variable können Sie den Wert direkt im `value`-Attribut von `progress` speichern.

Zum einfacheren Zugriff definieren Sie zunächst eine Funktion `progressbar`, die Ihnen das `progressbar`-Element zurückliefert.

```
const progressbar = () => $("#messages_progress");
```

Somit können Sie nun den aktuellen Wert des Fortschrittsbalken (`progressbar().value`) mittels `+=1` und `-=1` verändern.

```
1 const nextMessage = e => {
2   showMessageByNumber(progressbar().value += 1);
3   e.preventDefault();
4 };
5 const prevMessage = e => {
6   showMessageByNumber(progressbar().value -= 1);
7   e.preventDefault();
8 };
```

Ein kleines Refactoring

Zur Verbesserung der Lesbarkeit können Sie den Code, der den `value` der Fortschrittsbalken verändert, noch in eigene Funktionen auslagern:

```
1 const nextMessage = e => {
2   showMessageByNumber(incCurrentMessageNumber());
3   e.preventDefault();
4 };
5 const prevMessage = e => {
6   showMessageByNumber(decCurrentMessageNumber());
7   e.preventDefault();
8 };
9 const incCurrentMessageNumber = () => progressbar().value += 1;
10 const decCurrentMessageNumber = () => progressbar().value -= 1;
```

Probieren Sie es aus. Hurra — der Fortschrittsbalken funktioniert!

Initalisierung in progress

Um flexibel auf unterschiedlich große `messages`-Arrays reagieren zu können, sollten Sie den Fortschrittsbalken noch mit den richtigen Werten initialisieren.

```
1 const initProgressbar = () => {
2   progressbar().max = messages.length;
3   progressbar().value = 1;
4 };
```

`initProgressbar` können Sie dann in der `init`-Hauptfunktion aufrufen. Fügen Sie versuchshalber ein paar neue Nachrichten im Array hinzu. Sie sehen, dass der Fortschrittsbalken sich sofort dynamisch anpasst.

7.3 Achtung! — boolesche Attribute

Bei booleschen Attributen müssen Sie etwas aufpassen. Betrachten Sie beispielsweise die Eigenschaft `disabled`. Diese Eigenschaft gibt es unter anderem auf den Elementen `button`, `input`, `option`, `select` und `textarea`.

```
1 <button disabled="disabled">Not today, sorry</button>
2 <button>Click Me - I'm enabled</button>
```

Mit `disabled="disabled"` können Sie den Button deaktiveren. Er erscheint ausgegraut und lässt sich nicht mehr anklicken.

Abb. 7.1 *nicht-aktiver und aktiver Button*

Das IDL-Attribut hat den Typ `boolean`:

```
typeof $('button').disabled // => boolean
```

Sie können es mittels

```
$('button').disabled = true
```

bzw.

```
$('button').disabled = false
```

umschalten. Verwenden Sie Codebeispiel 7.3 und die Konsole, um es auszuprobieren.

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="utf-8" />
6    <title>Buttons</title>
7  </head>
8
9  <body>
10   <button disabled="disabled">Not today, sorry</button>
11   <button>Click Me - I'm enabled</button>
12 </body>
13
14 </html>
```

Codebeispiel 7.3 *additional_files/07/examples/button.html*

Arbeiten Sie jedoch mit den Content-Attributen, ist die korrekte Schreibweise zum Ausgrauen des Buttons `$('button').setAttribute('disabled', 'disabled')`. Noch verrückter wird es, wenn Sie versuchen, den Button wieder zu aktivieren: Ein naives `$('button').setAttribute('disabled', '')` hilft nicht.

Tatsächlich würde es den Wert von `disabled` auch auf `true` setzen — letztendlich geht es nämlich bei booleschen Attributen nur um die Frage, ob das Attribut vorhanden ist oder nicht. Der tatsächliche String-Wert spielt keine Rolle. Wenn Sie das Content-Attribut wieder loswerden möchten, müssen Sie es mit `removeAttribute` entfernen:

```
$('button').removeAttribute('disabled');
```

7.4 Auf die Fragestellung kommt es an — mit dem value-Attribut

Betrachten Sie Codebeispiel 7.4. Das Eingabefeld hat den vorgegebenen Text *hello* im `value`-Attribut.

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="utf-8" />
6    <title>Input</title>
7  </head>
8
9  <body>
10   <input type="text" value="hello" />
11 </body>
12
13 </html>
```

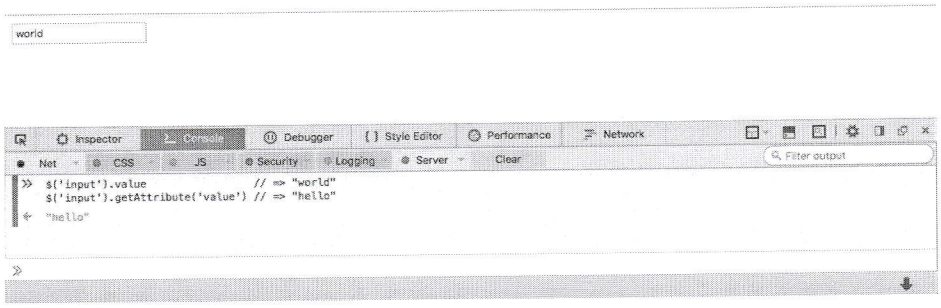
Codebeispiel 7.4 *additional_files/07/examples/input.html*

Solange Sie den Text nicht ändern, sind IDL- und Content-Attribut identisch.

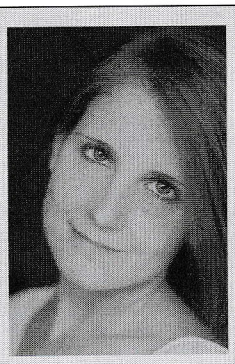
```
1 $('input').value                // => "hello"
2 $('input').getAttribute('value') // => "hello"
```

Geben Sie nun einen neuen Text in das Feld ein, z. B. *world*, und fragen Sie den Wert erneut über JS ab:

```
1 $('input').value                // => "world"
2 $('input').getAttribute('value') // => "hello"
```



Das IDL-Attribut reflektiert die Änderung und liefert somit die aktuelle Eingabe (hier: `"world"`). Das Content-Attribut bezieht sich aber nach wie vor auf den Wert aus dem HTML-Quellcode (hier: `"hello"`). Dieser Wert kann hier auch als *Standardwert* bezeichnet werden, da er für den Anwender bereits vorab im Input-Element vorgegeben ist.



Meine Empfehlung

Verwenden Sie immer die IDL-Attribute — außer Sie benötigen gezielt einen Originalwert aus dem HTML-Code!

Der Zugriff auf die IDL-Attribute ist aus mehreren Gründen vorzuziehen:

- Sie erhalten einen genaueren Datentyp und nicht nur String.
- Sie erhalten aktuelle, möglicherweise geänderte Werte (z. B. durch Eingaben).
- Das Verhalten ist konsistenter. Sie müssen weniger Sonderfälle beachten.

7.5 Zusammenfassung

Name des Attributs	Elemente, für die das Attribut oft verwendet wird	Beschreibung
alt	`area, img, input`	Alternativer Text, z. B. falls ein Bild nicht dargestellt werden kann.
checked	`input`	Gibt an, ob ein Radiobutton gewählt oder eine Checkbox angehakt ist.
cols	textarea	Anzahl der Spalten
disabled	`button, input, option, select, textarea`	Gibt an, ob ein Element deaktiviert ist.
href	`a, link`	Die URL einer verlinkten Ressource
max	`progress, input`	Der Maximalwert — bei `input`-Elementen ist das nur für `type="number"` und `type="range"` sinnvoll.
maxlength	`input, textarea`	Maximal erlaubte Zeichen in einem Element
min	`input`	Der Minimalwert — bei `input`-Elementen ist das nur für `type="number"` und `type="range"` sinnvoll.
multiple	`input, select`	Erlaubt bei `true` mehrere Eingaben für `input type="email"` oder `input type="file"`. Bei `select` können mit **strg/cmd** mehrere Optionen gewählt werden.

Tabelle 7.1 *Ein Auswahl von Attributen, die in der Praxis öfter mit JS angesprochen werden. Eine vollständige Liste aller Attribute finden Sie in der MDN Attribut reference*[36].

36. *https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes*

Name des Attributs	Elemente, für die das Attribut oft verwendet wird	Beschreibung
name	`button`, `form`, `fieldset`, `input`, `select`, `textarea`	Name des Elements. Dieses Attribut wird hauptsächlich für serverseitige Anwendungen zur Identifizierung genutzt.
placeholder	`input`, `textarea`	Voreingestellter Text, der dem Anwender einen Hinweis zu den erwarteten Eingaben gibt.
readonly	`input`, `textarea`	Gibt an, ob ein Element bearbeitet werden kann.
rel	`a, link`	Definiert die Beziehung zwischen dem Ziel- und dem Link-Objekt. Kann auch für eigene Beziehungstypen verwendet werden, z. B. interne Links für eine JS-Bildergalerie.
required	`input`, `select`, `textarea`	Gibt an, ob ein Element ein Pflichtfeld darstellt.
rows	`textarea`	Anzahl der Zeilen
selected	`option`	Gibt an, ob eine `option` selektiert ist.
size	`select`	Anzahl der sichtbaren Optionen bei einer `select`-Box
src	`img`	URL der Bilddatei
start	`ol`	Definiert die Startnummer einer Liste, wenn etwas anderes als 1 gewünscht ist.
step	`input`	Die Schrittweite — bei `input`-Elementen ist das nur für `type="number"` und `type="range"` sinnvoll.

Tabelle 7.1 *Ein Auswahl von Attributen, die in der Praxis öfter mit JS angesprochen werden. Eine vollständige Liste aller Attribute finden Sie in der MDN Attribut reference.*

Name des Attributs	Elemente, für die das Attribut oft verwendet wird	Beschreibung
tabindex	auf allen Elementen möglich	Überschreibt die herkömmliche Reihenfolge der angesteuerten (Formular-)Elemente mit Hilfe der Tab-Taste.
title	auf allen Elementen möglich	Kleine Text-Box (Tooltip), die beim »Hover« angezeigt wird.
type	`button`, `input`	Typ des Elements
value	`button`, `option`, `input`, `progress`	Wert des Elements

Tabelle 7.1 *Ein Auswahl von Attributen, die in der Praxis öfter mit JS angesprochen werden. Eine vollständige Liste aller Attribute finden Sie in der MDN Attribut reference.*

7.6 Übungen

Übung 12: Jetzt hat es sich »ausgebuyt«!

Anwender sind häufig ungeduldig und neigen dazu, mehrmals auf etwas zu klicken, wenn nicht sofort etwas Sichtbares passiert. Damit der Käufer die neue Heimkino-Anlage oder das Segelboot nicht versehentlich mehrfach bestellt, hat es sich bewährt, Buttons zu deaktivieren, nachdem sie geklickt wurden.

Schreiben Sie eine entsprechende Funktion für den im Begleitmaterial (*buy_button.html*) hinterlegten Button.

Übung 13: Newsboard: Jetzt ist aber Schluss!

Die neuen Features des Newsboards sind schon fast fertig. Problematisch ist aber, dass die Vorwärts- und Rückwärts-Buttons auch aktiv sind, wenn es keine vorige bzw. nächste Nachricht mehr gibt. Dies führt zu einem sehr unschönen Fehler.

1. Machen Sie die Links (a-Element), die der Steuerung der Newsboards dienen, zu echten Buttons (button-Element). Damit lassen sie sich im Bedarfsfall deaktivieren.
2. Sorgen Sie dafür, dass die Vor- und Rückwärts-Buttons jeweils deaktiviert werden, sobald der Anwender am Anfang bzw. Ende der Nachrichtenliste angekommen ist.
3. Auch die Buttons, die den Anwender zur ersten bzw. zur letzten Nachricht führen, sollen deaktiviert sein, sobald sich der Anwender bei der ersten bzw. letzten Nachricht befindet.

Übung 14: Die Rückkehr der Lauflichter: Teil 1 - Chasing Lights

Abb. 7.2 *Photo: Axel Kuhlmann*[37]
Lizenz: CC-BY-2.0[38]

Lichterketten waren einmal der letzte Schrei (und sind es um Weihnachten oft immer noch). Lauflicht-Funktion verfügen.

Genau so eine könnten Sie jetzt mal programmieren. Die folgende, einfache HTML-Vorlage finden Sie auch im Übungsmaterial:

37. *https://www.flickr.com/photos/abbilder/4921660612/*
38. *https://creativecommons.org/licenses/by/2.0/*

```
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
    <title>Beispiel</title>
    <meta http-equiv="content-type" content="text/html;
charset=utf-8" />
    <!-- <script src="running_light.js"
defer="defer"></script> -->
    <script src="../../../lib/dom_helper.js"></script>
</head>

<body>

<img src="light_on.png" alt="lightbulb" />
<img src="light_off.png" alt="lightbulb" />
<img src="light_off.png" alt="lightbulb" />
<img src="light_off.png" alt="lightbulb" />
<img src="light_off.png" alt="lightbulb" />
<img src="light_off.png" alt="lightbulb" />
<img src="light_off.png" alt="lightbulb" />
<img src="light_off.png" alt="lightbulb" />
<img src="light_off.png" alt="lightbulb" />
<img src="light_off.png" alt="lightbulb" />

</body>
</html>
```

Codebeispiel 7.7 *additional_files/07/exercises/running_light/running_light.html*

Eine erste Variante sind die *chasing Lights*. Dabei muss der Anwender das Lauflicht mit der Maus »jagen«. Immer wenn der Mauszeiger sich über einer Glühbirne befindet, schalten Sie die nächste Birne ein und die aktuelle aus. Versucht der Benutzer, mit dem Mauszeiger die leuchtende Glühbirne zu berühren, springt das Licht auf die nächste über. Er »jagt« quasi das Lauflicht.

Bei der letzten Lampe beginnt das Spiel wieder von vorne!

Übung 15: Die Rückkehr der Lauflichter: Teil 2 - Triggered Classic

Programmieren Sie ein weiteres Lauflicht. Dieses mal soll das Lauflicht weiterspringen, wenn der Benutzer auf irgendeine der Lampen klickt.

Auch hier gilt: Nach der letzten Lampe fängt das Licht wieder von vorne an zu laufen.

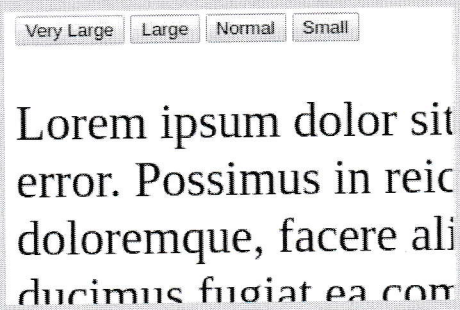
8 Eine Frage des Stils: Das Style-Objekt

Hey, wir haben gerade ein neue hippe Geschäftsidee gelaunched: Es gibt jetzt Nerd-Produkte für Senioren. Ich bin sicher, das wird der Renner!

Ich habe aber schon von meiner Oma und einigen älteren Herrschaften gehört, dass ihnen die Schriftgröße auf unserer Seite zu klein ist. Wir werden sie natürlich grundsätzlich vergrößern. Die Bedürfnisse sind nur leider recht unterschiedlich. Deswegen wäre es gut, wenn Sie ein paar Buttons auf die Seite werfen könnten, mit denen sich die Schriftgröße wählen lässt:

- small (14px)
- normal (16px)
- large (24px)
- very large (36px)
- »Mann, bin ich blind« (72px)
- ultra enormously extraordinary extremly large (200px)

Naja, vielleicht lassen Sie die letzten beiden erst einmal weg …



Sie könnten auch hier wieder CSS-Klassen für die verschiedenen Schriftgrößen anlegen, das wäre in diesem Fall aber übertrieben. Einfacher ist es, Sie manipulieren die Schriftgröße direkt.

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="utf-8" />
6   <title>Font Size</title>
7   <script src="../../../lib/dom_helper.js" defer="defer"></script>
8   <script src="font_size_refactored.js" defer="defer"></script>
9 </head>
10
11 <body>
12
13   <button id="very_big">Very Large</button>
14   <button id="big">Large</button>
15   <button id="normal">Normal</button>
16   <button id="small">Small</button>
17   <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
Nam, error. Possimus in reiciendis quo numquam assumenda, deleniti
doloremque, facere alias odio animi est aliquam delectus corrupti
ducimus fugiat ea commodi.</p>
18
19 </body>
20
21 </html>
```

Codebeispiel 8.1 *additional_files/08/examples/font_size_big_small/font_size.html*

Das DOM stellt dafür die Eigenschaft `style` auf den HTML-Element-Objekten zur Verfügung. Öffnen Sie Codebeispiel 8.1 im Browser. Geben Sie `$('p').style` in die Konsole ein —Sie erhalten ein `CSSStyleDeclaration`-Objekt zurück. Dieses Objekt stellt alle verfügbaren CSS-Eigenschaften als JS-Eigenschaften zur Verfügung, beispielsweise die Schriftfarbe `color`. Geben Sie ein:

```
$('p').style.color = 'red'
```

Der Text des ersten Absatzes (`p`-Tag) wird nun rot. Die Schriftgröße können Sie genauso verändern. Dabei gibt es nur eine kleine Spitzfindigkeit zu beachten.

TODO CE: Screenshot

Simsala-Bim: Du bist jetzt ein Kamel

In CSS haben viele Eigenschaften einen Bindestrich im Eigenschaftsnamen, so auch `font-size`. Das ist für JS-Bezeichner aber nicht zulässig. Deswegen müssen Sie die Schreibweise stattdessen in *camelCase* umwandeln: Aus `font-size` wird `fontSize`. Um die Schriftgröße zu vergrößern, können Sie eingeben:

```
$('p').style.fontSize = '36px'
```

Vergessen Sie die Einheit (hier: `px`) nicht, sonst ignoriert der Browser die Angabe!

Jetzt müssen Sie die Anweisungen nur noch an das `click`-Event des jeweiligen Buttons binden, z. B. so für den Big-Button:

```
$("#big").on("click", () => $("p").style.fontSize = "36px");
```

Codebeispiel 8.2 zeigt den vollständigen Code für alle Buttons.

```
1  "use strict";
2
3  {
4    const init = () => {
5      $("#very_big").on("click", () => $("p").style.fontSize =
"36px");
6      $("#big").on("click", () => $("p").style.fontSize = "24px");
7      $("#normal").on("click", () => $("p").style.fontSize = "16px");
8      $("#small").on("click", () => $("p").style.fontSize = "14px");
9    };
10
11   init();
12 }
```

Codebeispiel 8.2 *additional_files/08/examples/font_size_big_small/font_size.js*

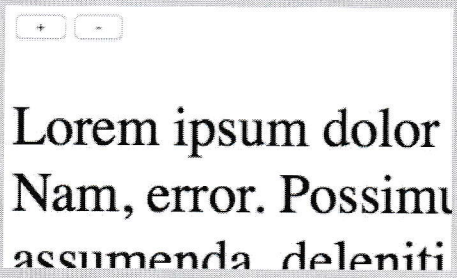
Das übliche Refactoring — hier das Entfernen von Redundanzen und Magic Numbers — führt zu folgendem Code.

```
1  "use strict";
2
3  {
4    const SMALL = 14;
5    const NORMAL = 16;
6    const BIG = 24;
7    const VERY_BIG = 36;
8
9    const init = () => {
10     $("#very_big").on("click", () => setFontSizeTo(VERY_BIG));
11     $("#big")      .on("click", () => setFontSizeTo(BIG));
12     $("#normal")   .on("click", () => setFontSizeTo(NORMAL));
13     $("#small")    .on("click", () => setFontSizeTo(SMALL));
14   };
15
16   const setFontSizeTo = size =>  $("p").style.fontSize = size +
"px";
17
18   init();
19 }
```

Codebeispiel 8.3 *additional_files/08/examples/font_size_big_small/font_size_refactored.js*

8.1 Absolut berechnend: getComputedStyle

Oh — hmm ja — also. Sie haben das ja schon toll umgesetzt, aber wie soll ich das sagen: Wir müssen es nochmal ändern. Wir haben Umfragen bei unseren Kunden durchgeführt und festgestellt, dass sie lieber auf »+« und »–«-Buttons zur Änderung der Schriftgröße klicken würden.



Kein Problem für Sie — hier ist das neue HTML-Dokument:

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <title>Font Size</title>
  <script src="../../../lib/dom_helper.js" defer="defer"></script>
  <script src="font_size.js" defer="defer"></script>
</head>

<body>

  <button id="inc">+</button>
  <button id="dec">–</button>
  <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.
Nam, error. Possimus in reiciendis quo numquam assumenda, deleniti
doloremque, facere alias odio animi est aliquam delectus corrupti
ducimus fugiat ea commodi.</p>

</body>

</html>
```

Codebeispiel 8.4 *additional_files/08/examples/font_size_inc_dec_non_functional/font_size.html*

Registrieren Sie zunächst wieder die Click-Events auf den Buttons. Sinnvoll ist es, die Funktionen zum Vergrößern (engl. *increase*) und Verkleinern (engl. *decrease*) der Schriftgröße entsprechend `incFontSize` und `decFontSize` zu nennen.

```
1 const init = () => {
2   $("#inc").addEventListener("click", incFontSize);
3   $("#dec").addEventListener("click", decFontSize);
4 };
```

Legen Sie die beiden Funktionen an — erst einmal `incFontSize`.

Die Idee ist ganz einfach. Sie greifen auf die aktuelle Schriftgröße zu. Wir tun einfach mal so, als ob es bereits eine Funktion namens `currentFontSize` gäbe, die uns die Schriftgröße als `number` zurückliefert. Zu diesem Wert addieren Sie `5`. Das Ergebnis ist die neue `fontSize`, die Sie wieder dem Style-Objekt zuweisen.

```
const incFontSize = () => setFontSizeTo(currentFontSize() + 5);
```

Mit dem Verkleinern verfahren Sie analog, indem Sie `5` von der aktuellen Schriftgröße (`currentFontSize`) abziehen:

```
1 const decFontSize = () => setFontSizeTo(currentFontSize() - 5);
```

Damit das auch funktioniert, müssen Sie nur noch die Funktion `currentFontSize` implementieren. Hier ein erster Versuch

```
1 const currentFontSize = () => parseInt($("p").style.fontSize);
```

Die Idee ist hier, die bestehende `fontSize` auszulesen und per `parseInt` eine Zahl ohne das `"px"` am Ende zu extrahieren.

parseInt oder das Geheimnis der verschwundenen Pixel

Am besten probieren Sie erst einmal `parseInt` direkt in der Konsole aus. Die Funktion ist hier außerordentlich nützlich, da sie Ihnen erlaubt, das `"px"` am Ende der `fontSize` loszuwerden. Geben Sie z.B. `"40px" - 5` in der Konsole ein, erhalten Sie `NaN` (Not a Number) als Rückgabewert. Der String `"40px"` ist nun mal keine Zahl, damit lässt sich leider nicht rechnen.

Mittels `parseInt` können Sie die Zahl extrahieren. Die Funktion nimmt einen String entgegen und ***parst*** diesen, soweit sie gültige Zeichen (d.h. Ziffern) findet. Geben Sie in die Konsole ein:

```
parseInt("40px")
```

Sie erhalten `40` als *Number* zurück — damit können Sie rechnen.

Das erste Problem ist bereits gelöst. Betrachten Sie nun nochmal die Implementierung von `currentFontSize`:

```
1 const currentFontSize = () => parseInt($("p").style.fontSize);
```

Leider funktioniert der Code immer noch nicht! Woran liegt das?

Wirklich eine Frage des Stils?

Lassen Sie uns ein kleines Experiment machen. Versuchen Sie mal, einen bestehenden Style auszulesen. Laden Sie die Seite neu und öffnen Sie die JS-Konsole. Geben Sie jetzt das Style-Objekt aus, indem Sie folgende Zeile in Konsole eintippen:

```
console.log($('p').style)
```

Sie sehen, das Style-Objekt hat den Typ `CSSStyleDeclaration`. Klappen Sie es aus und zeigen Sie es an.

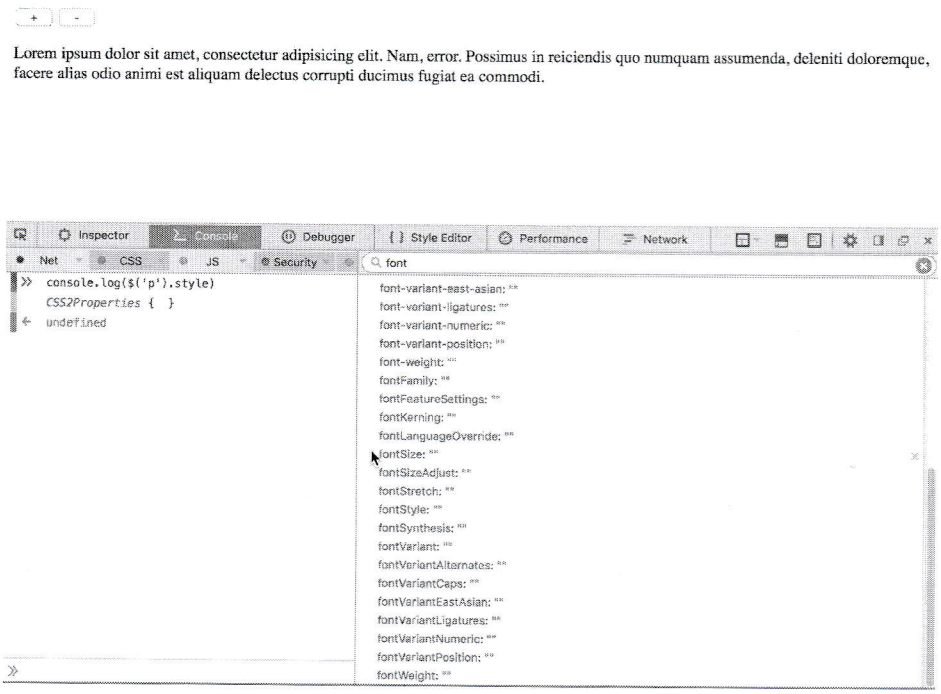


Abb. 8.1 *`fontSize`-Eigenschaft im `style`-Objekt mit Leerstring*

Es hat eine Menge Eigenschaften. Die `fontSize` ist auch dabei. Das Erstaunliche ist aber, dass diese Eigenschaft statt einer tatsächlichen FontSize wie `"16px"` nur den leeren String `""` enthält.

Weisen Sie der `fontSize` versuchsweise den Wert "40px" zu — wieder in der Konsole.

```
$("p").style.fontSize = "40px";
```

Geben Sie es erneut aus.

```
console.log($('p').style);
```

Dieses Mal finden Sie die `"40px"` tatsächlich als Wert der `fontSize` im Style-Objekt.

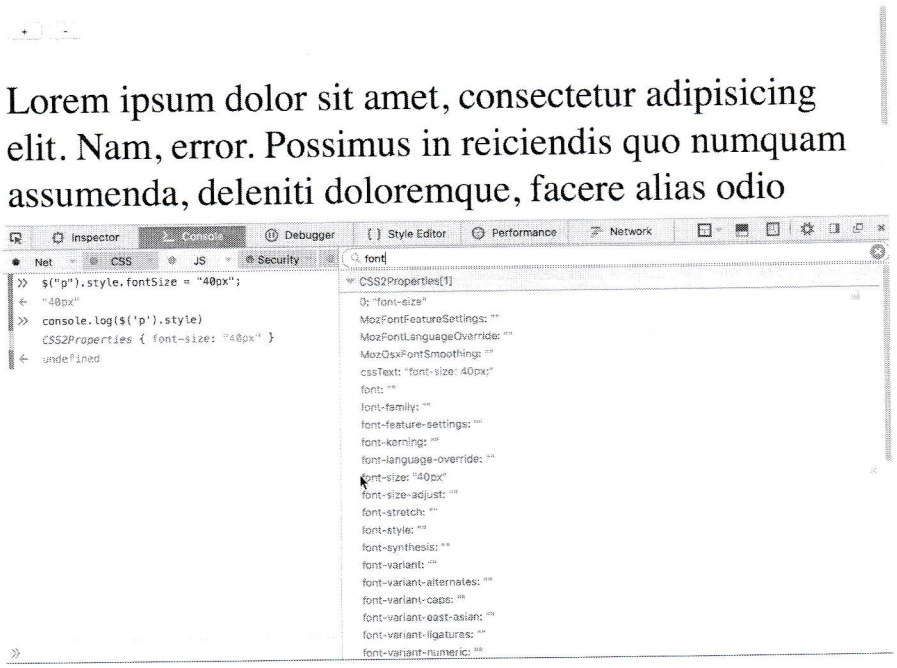


Abb. 8.2 *`fontSize`-Eigenschaft im `style`-Objekt mit vorhandenem px-Wert*

Genau darin besteht das Problem: **Sie können einen Wert erst dann aus dem Style-Objekt auslesen, wenn Sie ihn vorher mit JS gesetzt haben**. Auch ein sogenannter Inline-Style (style-Attribut) würde den Wert im Objekt setzen. In der Praxis empfiehlt es sich aber nicht, mit Inline-Styles zu arbeiten. Standwerte, die der Browser setzt oder Werte, die Sie in einem CSS-Stylesheet festelegt haben, gibt das Style-Objekt leider nicht preis.

Styling-Werte grundsätzlich mit JS statt mit CSS festzulegen, wäre aber auch keine gute Lösung — Nerdworld-Designerin Denise würde das gar nicht gut finden …

Grundsatzfragen mit getComputedStyle

Tatsächlich gibt es eine gute Lösung: die Funktion `getComputedStyle`. Diese Funktion ist eine Methode des `window`-Objektes und dadurch global verfügbar

— d.h. statt `window.getComputedStyle` genügt es, einfach nur `getComputedStyle` zu schreiben. Sie steht in allen modernen Browsers zu Verfügung und ist im IE ab Version 9 nutzbar.

Übergeben Sie `getComputedStyle` ein HTML-Element-Objekt, wie z.B. unser `p`-Tag, erhalten Sie ein Style-Objekt zurück — wie gewohnt vom Typ `CSSStyleDeclaration`. Das Besondere daran ist aber, dass dieses Objekt die tatsächlichen, berechneten Werte beinhaltet. So wertet die Funktion auch Browserstandards und CSS-Eigenschaften inkl. externer Stylesheets aus. Probieren Sie es: Laden Sie die Seite neu und geben Sie ein:

```
console.log(getComputedStyle($("p")).fontSize)
```

Sie erhalten `16px`, den Standardwert des Browsers.

Wenn Sie sich das komplette Objekt ausgeben, sehen Sie, dass praktisch alle Eigenschaften mit Werten belegt sind.

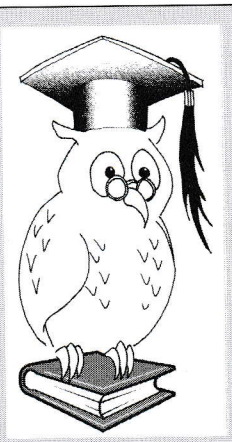
Perfekt — da wird Björn sich aber freuen. Sie können `getComputedStyle` nun für Ihre Funktion `currentFontSize` verwenden:

```
const currentFontSize = () =>
  parseInt(getComputedStyle($("p")).fontSize);
```

Hier ist der komplette Code:

```
1  "use strict";
2
3  {
4    const init = () => {
5      $("#inc").on("click", incFontSize);
6      $("#dec").on("click", decFontSize);
7    };
8
9    const incFontSize = () => setFontSizeTo(currentFontSize() + 5);
10   const decFontSize = () => setFontSizeTo(currentFontSize() - 5);
11
12   const currentFontSize = () =>
13     parseInt(getComputedStyle($("p")).fontSize);
14
15   const setFontSizeTo = size =>
16     $("p").style.fontSize = size + "px";
17
18   init();
19 }
```

Codebeispiel 8.5 *additional_files/08/examples/font_size_inc_dec/font_size.js*



Besonderheiten und Nachteile von getComputedStyle

Tja, das hört sich ja schon echt gut an mit dem `getComputedStyle`. Auch, wenn ich nur ungern der Spaßverderber sein will, **aber**: Zwei Sachen muss ich klarstellen.

Read-Only

Das `CSSStyleDeclaration`-Objekt, das da zurückkommt, unterscheidet sich in einem ganz wesentlichen Aspekt vom normalen `.style`-Objekt: Die Eigenschaften lassen sich nicht ändern! Sie sind ***Read-Only***. Das ist nicht wirklich ein Problem, schließlich lässt sich dafür das `.style`-Objekt verwenden — aber Sie müssen daran denken.

Geben Sie z. B.

```
getComputedStyle($("p")).fontSize = "19px";
```

in die Konsole ein, straft Sie der Browser mit Meldungen wie dieser:

```
VM36020:1 Uncaught DOMException: Failed to set
the 'font-size' property on
'CSSStyleDeclaration': These styles are
computed, and therefore the 'font-size'
property is read-only.(…)
```

Perfomance

`getComputedStyle` ist nicht gerade dafür bekannt, ein Rennpferd zu sein. Es ist viel, viel langsamer als `.style`. Deswegen ist es manchmal ein guter Trick, den ersten Wert per `getComputedStyle` zu ermitteln (z. B. in der `init`-Funktion) und danach mit dem deutlich schnelleren `.style` zu arbeiten.

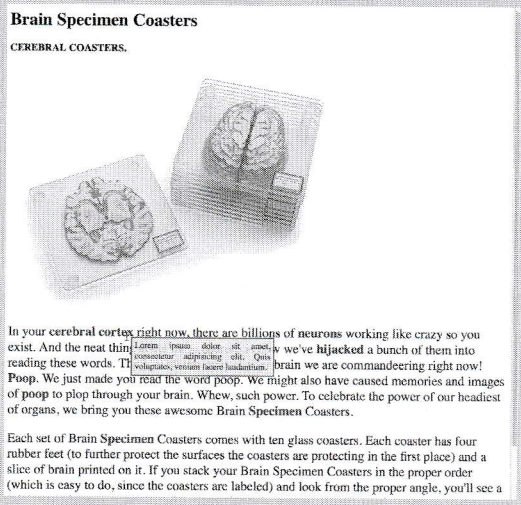
8.2 Noch style-ischer mit Tooltips

Wow — ich hab gestern Abend meiner Oma unseren neuen Prototyp gezeigt! Dank »+«-Button konnte sie den Text ohne Brille lesen, sie war ganz begeistert.

Allerdings konnte sie mit einigen Begriffen auf der Website wenig anfangen und wollte wissen, was *cerebral cortex* und *neuron* eigentlich heißt. Könnten wir vielleicht kleine Erklärungstooltips einblenden, wenn jemand mit der Maus über ein Fremdwort fährt?

Achso und nein — ich meine nicht diese Minitooltips, die man durch das `title`-Attribut bekommt, sondern schöne große Tooltip-Boxen, die Denise, unsere Designerin, auch stylen kann.

Die Erklärungstexte zu den Begriffen sollen später über eine externe Datei eingelesen werden — so, dass wir sie selbst pflegen können. Aber das brauchen wir nicht sofort. Wir schicken Ihnen die Texte per E-Mail und Sie können sie dann direkt in den Code einbinden.



Brain Specimen Coasters

CEREBRAL COASTERS.

In your **cerebral cortex** right now, there are billions of **neurons** working like crazy so you exist. And the neat thing, we've **hijacked** a bunch of them into reading these words. The brain we are commandeering right now! **Poop**. We just made you read the word poop. We might also have caused memories and images of **poop** to plop through your brain. Whew, such power. To celebrate the power of our headiest of organs, we bring you these awesome Brain **Specimen** Coasters.

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Quis voluptates, veniam facere laudantium.

Each set of Brain **Specimen** Coasters comes with ten glass coasters. Each coaster has four rubber feet (to further protect the surfaces the coasters are protecting in the first place) and a slice of brain printed on it. If you stack your Brain Specimen Coasters in the proper order (which is easy to do, since the coasters are labeled) and look from the proper angle, you'll see a

Los gehts! Nach kurzem Überlegen wird klar: Das ist wieder ein Fall für das Style-Objekt. Betrachten Sie folgende HTML-Vorlage, die eine vereinfachte Fassung der echten Seite[39] darstellt:

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8" />
6   <title>Tooltips</title>
7
8   <script src="../../../../lib/dom_helper.js"
defer="defer"></script>
9   <script src="tooltips.js" defer="defer"></script>
10
11  <link rel="stylesheet" href="tooltips.css" />
12 </head>
13
14 <body>
15   <h1>Brain Specimen Coasters</h1>
16   <h3>CEREBRAL COASTERS.</h3>
17
18   <img src="../images/brain_specimen_coasters.jpg"
alt="brain_specimen_coasters" />
19
20   <p>In your <span class="keyword">cerebral cortex</span> right
now, there are billions of <span class="keyword">neurons</span>
working like crazy so you exist. And the neat thing about these
neurons is how we've <span class="keyword">hijacked</span> a bunch of
them into reading these words. Think of all the areas of your brain
we are commandeering right now! <span class="keyword">Poop</span>. We
just made you read the word poop. We might also have caused memories
and images of <span class="keyword">poop</span> to plop through your
brain. Whew, such power. To celebrate the power of our headiest of
organs, we bring you these awesome Brain <span
class="keyword">Specimen</span> Coasters.</p>
21
22   <p>Each set of Brain <span class="keyword">Specimen</span>
Coasters comes with ten glass coasters. Each coaster has four rubber
feet (to further protect the surfaces the coasters are protecting in
the first place) and a slice of brain printed on it. If you stack
your Brain Specimen Coasters in the proper order (which is easy to
do, since the coasters are labeled) and look from the proper angle,
you'll see a full brain. Plus, if you use your Brain Specimen
Coasters at your next science party, your friends will know how
thoughtful you really are. Get it? Brain Specimen Coasters?
Thoughtful? Uhhh.... <span class="keyword">poop</span>.</p>
23 </body>
24
25 </html>
```

Codebeispiel 8.6 *additional_files/08/examples/tooltips/1/tooltips.html*

Björn hat einige der Wörter mit der Klasse `keyword` markiert. Das sind die Wörter, die ein Tooltip mit Erläuterung benötigen. Diese Wörter müssen Sie selektieren. Probieren Sie zunächst folgende Anweisung in der JS-Konsole aus:

39. Herzlichen Dank für die freundliche Genehmigung an Think Geek Inc.! Entnommen aus *www.thinkgeek.com/product/huir/*

```
$$(".keyword").forEach(n => console.log(n.innerHTML));
```

Sie erhalten:

```
cerebral cortex
neurons
hijacked
Poop
poop
Specimen
poop
```

So weit, so gut! Erweitern Sie die Zeile nun zu einem richtigen Programm und registrieren Sie eine Funktion `showTooltip` auf das `mouseenter`-Event. Der Browser löst dieses Event aus, sobald sich der Mauszeiger über eines der Wörter bewegt.

Im ersten Schritt möchten wir einfach nur sehen, ob die Funktion richtig angebunden ist. Deswegen genügt auch hier eine Ausgabe mit `console.log`.

```
1 "use strict";
2
3 {
4   const init = () => $$(".keyword").on("mouseenter", showTooltip);
5
6   const showTooltip = e => console.log(e.target.innerHTML);
7
8   init();
9 }
```

Codebeispiel 8.7 *additional_files/08/examples/tooltips/1/tooltips.js*

Als Nächstes benötigen Sie das Tooltip. Zum Glück hat Nerdworlds Designerin Denise bereits eins erstellt und mit CSS gestylt. Das Tooltip selbst ist lediglich ein `div`-Tag, der den Erklärungstext enthält — da Björn noch keine Texte geliefert hat, reicht vorerst auch ein *lorem ipsum*-Platzhaltertext.

```
1 <div id="tooltip">Lorem ipsum dolor sit amet, consectetur
adipisicing elit. Quis voluptates, veniam facere laudantium.</div>
```

Fügen Sie den Tag einfach am Ende des HTML-Dokuments vor dem schließenden `body`-Tag ein.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>...</head>
4 <body>
5   <p>...</p>
```

```
6   <div id="tooltip">Lorem ipsum dolor sit amet, consectetur
adipisicing elit. Quis voluptates, veniam facere laudantium.</div>
7 </body>
8 </html>
```

Dank der folgenden CSS-Regel aus *tooltips.css* besitzt das Tooltip bereits eine ansprechende Optik:

```
1 #tooltip {
2   width: 250px;
3   border: 2px solid #339;
4   padding: 5px;
5   background-color: #eef;
6   opacity: 0.95;
7   font-size: 1em;
8   text-align: justify;
9 }
```

Denise wird das später sicher noch schöner machen.

Fügen Sie der Regel den Stil `display: none;` hinzu, damit das Tooltip nach dem Laden der Seite zunächst versteckt ist. Erst wenn ein Besucher der Website den Mausezeiger über eines der Wörter bewegt, soll sich das Tooltip zeigen. Dazu müssen Sie lediglich per JavaScript den `display`-Wert des Tooltips auf `block` setzen. Das liegt daran, dass es sich dabei um ein `div`-Element handelt. Bei einem `span`-Element müssten Sie stattdessen den Wert `inline` verwenden.

```
$("#tooltip").style.display = "block";
```

Das Setzen des Display-Stils ist die Aufgabe der Funktion `showTooltip`. Modifizieren Sie sie entsprechend.

```
const showTooltip = () => $("#tooltip").style.display = "block";
```

...und wieder weg

Das Tooltip sollte wieder verschwinden, wenn der Besucher den Mauszeiger wegbewegt. Dazu benötigen Sie noch eine passende Funktion `hideTooltip`, die Sie am besten auf das `mouseleave`-Event registrieren. Hier ist der komplette Code:

```
"use strict";

{
  const init = () => $$(".keyword")
    .on("mouseenter", showTooltip)
    .on("mouseleave", hideTooltip);

  const showTooltip = () => $("#tooltip").style.display = "block";
```

```
  const hideTooltip = () => $("#tooltip").style.display = "none";

  init();
}
```

Codebeispiel 8.8 *additional_files/08/examples/tooltips/2/tooltips.js*

Damit das Tooltip auch wirklich ein Tooltip ist, darf es nicht unter dem Text stehen, sondern sollte immer in der Nähe des Mauszeigers erscheinen. Gdereben Sie dem Tooltip im CSS die Eigenschaft `position: absolute`, um eine freie Positionierung zu ermöglichen:

```
1  p {
2    font-size: 1.5em;
3  }
4
5  .keyword {
6    font-weight: bolder;
7    color: #225;
8  }
9
10 #tooltip {
11   width: 250px;
12   border: 2px solid #339;
13   padding: 5px;
14   background-color: #eef;
15   opacity: 0.9;
16
17   font-size: 1em;
18   text-align: justify;
19
20   display: none;
21   position: absolute;
22 }
```

Codebeispiel 8.9 *additional_files/08/examples/tooltips/3/tooltips.css*

Zudem sollten Sie das Event von `mouseenter` auf `mousemove` ändern, damit sich die Position ständig anpasst:

```
$$(".keyword").on("mousemove", showTooltip) …
```

Maus im Fokus

Als Nächstes gilt es herauszufinden, wo sich der Mauszeiger eigentlich befindet. Bei Mouse-Events kümmert sich praktischerweise das Event-Objekt darum. Es zeichnet die aktuelle Position des Mauszeigers in den Eigenschaften `clientX` und `clientY` auf.

Erweitern Sie die Funktion `showTooltip` — so dass das Tooltip an die Position des Mauszeigers springt:

```
const showTooltip = e => {
  $("#tooltip").style.display = "block";
  $("#tooltip").style.top = e.clientY + "px";
  $("#tooltip").style.left = e.clientX + "px";
};
```

Mit den Style-Eigenschaften `top` und `left` lässt sich das Tooltip pixelgenau positionieren — möglicherweise etwas zu genau. Nun »klebt« das Tooltip immer genau am Mauszeiger. Ein geringer Abstand von z.B. `10px` behebt das Problem.

```
$("#tooltip").style.top = e.clientY + 10 + "px";
$("#tooltip").style.left = e.clientX + 10 + "px";
```

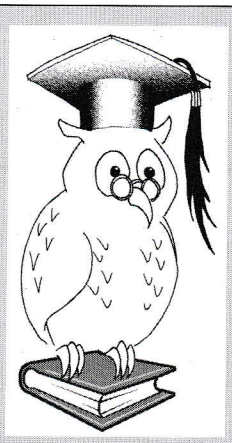
Am besten lagern Sie den Wert noch in eine Konstante aus, um die ***Magic Number***[40] loszuwerden. Hier der vollständige Code:

```
"use strict";

{
  const MOUSE_TO_TOOLTIP = 10;

  const init = () => {
    $$(".keyword").on("mousemove", showTooltip).on("mouseleave",
hideTooltip);
  };

  const showTooltip = e => {
    $("#tooltip").style.display = "block";
    $("#tooltip").style.top = e.clientY + MOUSE_TO_TOOLTIP + "px";
    $("#tooltip").style.left = e.clientX + MOUSE_TO_TOOLTIP + "px";
  };

  const hideTooltip = () => $("#tooltip").style.display = "none";

  init();
}
```

Codebeispiel 8.10 *additional_files/08/examples/tooltips/3/tooltips.js*

Hurra — die Tooltips funktionieren. Oma hat sich riesig gefreut. Endlich weiß sie, was der *cerebral cortex* ist. Naja, vielleicht hätten wir *poop* nicht erklären müssen…

40. Eine genaue Erklärung dazu finden Sie in *JavaScript Band 1 - Aller Anfang ist leicht*



Stilvoll oder eher mit Klasse?

Es gibt oft Situationen, in denen Sie eine Entscheidung treffen müssen. Entweder Sie arbeiten mit `classList.add/classList.remove`, um CSS-Eigenschaften zu verändern (siehe Abschnitt 3.2) — oder aber Sie greifen auf das Style-Objekt zu und verändern eine Eigenschaft direkt. Wann sollten Sie was tun?

In den allermeisten Fällen ist es besser, Sie arbeiten mit CSS-Klassen. Das hat den großen Vorteil, dass Sie Design-Aspekte (CSS) von Verhaltens-Aspekten (JS) getrennt halten. Innerhalb der CSS-Klasse sind die CSS-Eigenschaften gekapselt. Der JS-Code entscheidet nur über das Hinzufügen und Entfernen von Klassen, aber nicht darüber, welche konkreten Eigenschaften das betrifft.

So kann beispielsweise ein Designer hinterher das CSS verbessern, ohne den JS-Code anfassen zu müssen. Selbst wenn Designer und Entwickler die gleiche Person sind, erleichtert das die Arbeit ungemein. Sind JS und CSS-Code vermischt, bedeutet eine Änderung an der Optik (CSS) immer ein Risiko, auch unerwünschte Verhaltensänderungen (Bugs!) einzuführen.

Dennoch gibt es Situationen, in denen es besser ist, mit dem Style-Objekt direkt zu arbeiten — wie etwa im Beispiel mit den Schriftarten. Sie müssten dort für eine große Spanne von Fontgrößen jeweils passende Klassen anlegen, da die *Plus/Minus*-Buttons mit einem Delta von 5px arbeiten.

8.3 Übungen

Übung 16: 01001000010011110101010 0 — Teil 6

Fragen Sie besser immer noch nicht nach dem Sinn. Der neue Wunsch besteht darin, den Text blau (`blue`) zu färben. Dieses Mal dürfen Sie keine CSS-Klassen verwenden.

Weisen Sie allen `p`-Elementen mit Hilfe des Style-Objektes die Farbe `blue` zu. Sie können dafür direkt die JavaScript-Konsole verwenden.

Übung 17: TABula Rasa

Einer Ihrer Kunden ist ein Schulungsunternehmen. Der Akademieleiter wünscht sich, dass die Beschreibungsseiten der Seminare etwas übersichtlicher dargestellt werden. Er möchte deswegen, dass die Seminarübersichtsseiten in Zukunft über Reiter (*Tabs*) verfügen, so dass Besucher zwischen den Bereichen

- *Überblick*,
- *Themen*,
- *Lernziele* und
- *Termine*

hin- und herschalten können.

Eine HTML-Vorlage (inkl. CSS) hat er bereits von einer Agentur erstellen lassen. Ihre Aufgabe besteht darin, den JS-Code zu ergänzen.

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="utf-8" />
6
7    <title>SEO Seminar: 2-tägige Schulung
Suchmaschinenoptimierung</title>
8
9    <link href="tabs.css" media="all" rel="stylesheet"
type="text/css" />
10   <script src="../../../lib/dom_helper.js"
defer="defer"></script>
11   <script src="tabs.js" defer="defer"></script>
12 </head>
13
14 <body>
15
16   <div class="tabs">
17     <nav>
18       <ul>
19         <li>Überblick</li>
20         <li>Themen</li>
21         <li>Lernziele</li>
22         <li>Termine</li>
23       </ul>
```

```
24      </nav>
25
26      <article>
27        <h1>2-tägiges SEO-Intensiv-Seminar: Mit
Suchmaschinenoptimierung (SEO) zum Erfolg im Web!</h1>
28        <p>&hellip;</p>
29      </article>
30
31      <article>
32        <h2>Themen</h2>
33        <p>&hellip;</p>
34      </article>
35
36      <article>
37        <h2>Lernziele</h2>
38        <p>&hellip;</p>
39      </article>
40
41      <article>
42        <h2>Termine</h2>
43        <p>&hellip;</p>
44      </article>
45    </div>
46
47  </body>
48
49  </html>
```

Codebeispiel 8.11 *additional_files/08/exercises/tabbed_navigation/tabs.html*

Folgende Features müssen Sie dafür implementieren:

- Nach dem Laden müssen alle `article`-Elemente bis auf das erste versteckt sein.
- Beim Klicken auf ein `li`-Element soll das zugehörige `article`-Element aktiv (d.h sichtbar) werden und alle anderen inaktiv.
- Um zu sehen, welcher Tab gerade aktiv ist, darf außerdem nur das `li`-Element des gerade sichtbaren Tabs die CSS-Klasse `active` haben.

Übung 18: Farbe Wechsel Dich

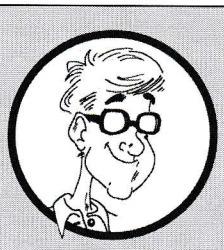
Platzieren Sie drei Buttons (`button`-Tag) auf einer Seite mit den Texten »Red«, »Green« und »Blue«. Bei Betätigung eines Buttons soll sich der Hintergrund der Seite entsprechend färben.

Übung 19: Farbe Wechsel Dich — Teil 2

Platzieren Sie drei Regler (`input type="range"`) auf einer Seite, die jeweils für einen Farbanteil des RGB-Farbmodells zuständig sind. Jedes mal, wenn ein Anwender einen Regler verschiebt, soll sich der Hintergrund der Seite entsprechend anpassen. Die drei Regler bestimmen dabei die Farbe anhand der entsprechenden Farbanteile.

HTML unterwandern mit Data-Attributen

9



Wie schon gesagt: Die Tooltips sind super! Wir benötigen jetzt aber ers tmal eine einfache Möglichkeit, die Texte der Tooltips zu pflegen. Mir wäre es am liebsten, wenn ich sie erst mal direkt im HTML hinterlegen könnte. Geht das irgendwie?

Ja, das geht. Es gibt sogar mehrere Möglichkeiten. Am einfachsten ist es, sogenannte `data-Attributes` zu verwenden. Damit sind Sie in der Lage, beliebige Inhalte quasi »unsichtbar« im HTML zu speichern!

HTML erlaubt es, eigene Attribute zu definieren, sofern diese mit `data-` beginnen. Sie können beispielsweise ein Attribut namens `data-tooltip` definieren und dort den jeweiligen Text hinterlegen. Das `tooltip` ist dabei ein Name, den Sie selbst wählen können. Er ist von HTML nicht vorgeschrieben. Sie haben dabei nahezu beliebige, aber nicht alle Freiheiten (z.B. Großbuchstaben sind nicht erlaubt). Auch Tooltips wie `data-foo`, `data-der-kunde-ist-doof` oder `data-was-immer-sie-wollen` wären möglich.

```
1 <p>In your <span class="keyword" data-tooltip="The brain's outer
layer of neural tissue in humans and other mammals
[wikipedia]."><cerebral cortex</span> right now,…
```

Versuchen Sie, das Attribut testweise in der Konsole auszulesen. Dazu benötigen Sie zunächst das Element: `$(".keyword")`. Um an ein `data-`-Attribute ranzukommmen, benutzen Sie die Eigenschaft `dataset`:

```
$(".keyword").dataset
```

Sie erhalten eine *DOMStringMap* zurück. Die können Sie behandeln wie jede andere Map. Jedes Data-Attribut ist in dieser Map mit seinem eigenen Key vertreten — so z.B. auch `tooltip`. Hier ist der Rückgabewert der letzten Anweisung:

```
1 DOMStringMap {tooltip: "The brain's outer layer of neural tissue in
humans and other mammals [wikipedia]."}
```

Jetzt müssen Sie Ihren JS-Code so anpassen, dass er statt des statischen Lorem Ipsums den korrekten Text aus dem `data`-Attribut zeigt. Fügen Sie die folgende Zeile in die Funktion `showTooltip` ein:

```
$("#tooltip").textContent = e.target.dataset.tooltip;
```

Das Attribut `textContent` ermöglicht es, einem Tag (wie hier dem `tooltip`-div, das das Lorem Ipsum enthält) einen beliebigen Textinhalt zuzuweisen. Damit ist es dem Attribut `innerHTML` recht ähnlich. Tatsächlich könnten Sie hier problemlos auch `innerHTML` verwenden. Wenn es Ihnen aber, wie hier, um reinen Text geht, ist `textContent` vorzuziehen. `textContent` interpretiert einen zugewiesenen String nicht als HTML — Sie können auch Zeichen wie `&` oder `<` direkt im Text verwenden.

9.1 Zusammenfassung

Attribut	Beispiel	Beschreibung
dataset	$("#bar").dataset	Enthält eine *DOMStringMap* mit den Daten der `data`-Attribute.
textContent	$("#bar").textContent	Gibt den Text (ohne HTML-Tags) zurück, der sich im HTML-Element befindet — der Wert kann auch verändert werden.

Tabelle 9.1 *Attribute dieser Lektion*

9.2 Übung

Übung 20: awesome tours

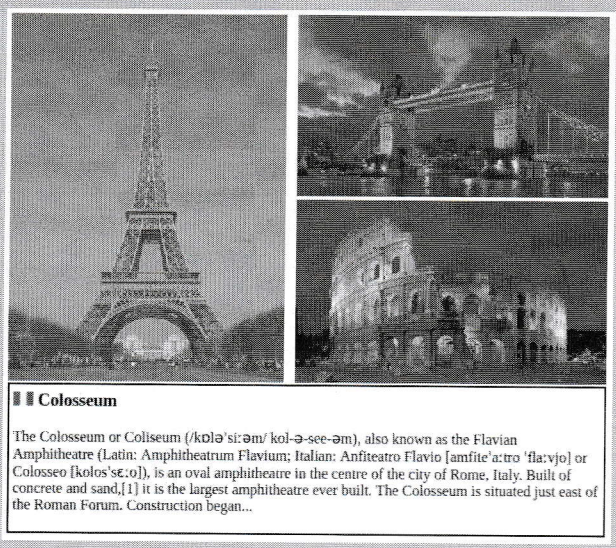


Abb. 9.1 *awesome tours Prototyp*

Das Reiseunternehmen *awesome tours* bietet Rundreisen zu verschiedenen Sehenswürdigkeiten in Europa an. Im Zuge eines Rebuilds der Website möchte das Unternehmen die Darstellung der Sehenswürdigkeiten ebenfalls modernisieren. Das Design inklusive HTML- und CSS-Code hat bereits eine Agentur übernommen.

```
1 ...
2 <body>
3   <section id="pois">
4     <img src="eiffelturm.jpg" id="fr" alt="Eiffel Tower"
data-description="The Eiffel Tower is a wrought iron lattice
tower on the Champ de Mars in Paris, France. It is named..."
data-country-code="fr" data-flag-name="French Flag"/>
5     <img src="tower_bridge.jpg" id="uk" alt="Tower Bride"
data-description="Tower Bridge (built 1886–1894) is a combined
bascule and suspension bridge in London. The bridge..."
data-country-code="uk" data-flag-name="Union Jack"/>
6     <img src="colosseum.jpg" id="it" alt="Colosseum"
data-description="The Colosseum or Coliseum, also known as the
Flavian Amphitheatre or Colosseo, is an oval amphitheatre in..."
data-country-code="it" data-flag-name="Italian Flag"/>
7   </section>
8   <section id="info"><p>Move your mouse pointer over the
images to read the descriptions here!</p></section>
9 </body>
10 ...
```

Die Informationen zu den Sehenswürdigkeiten (Name, Beschreibung, Land usw.) sind dabei bereits im HTML-Code hinterlegt. Ihre Aufgabe ist es nun-beim Mouseenter eine Darstellung wie in Abb. 9.1 zu ermöglichen.

Dazu gehört die Anzeige von:

- dem Namen der Sehenswürdigkeit
- der Beschreibung
- der Landesflagge als Bild.

Das HTML zur Anzeige der Eiffelturm-Infos könnte beispielsweise so aussehen:

```
1 <section id="info">
2   <h3>
3       <img src="it.png" alt="Italian Flag" title="Italian
Flag">
4       Colosseum
5   </h3>
6   <p>
7     The Colosseum or Coliseum, also known as the Flavian
Amphitheatre or Colosseo, is an oval amphitheatre in...
8   </p>
9 </section>
```

Von neuen Elementen und angehängten Kindern: createElement & appendChild

10

Die Mädels und Jungs von der Business-Development-Unit haben den Servercode für unser neues Produktmanager-Tool fertig. Wir sind uns aber noch unsicher, wie die Oberfläche dazu aussehen könnte. Deswegen bräuchten wir dringend mal einen Prototyp. Der muss noch keine Verbindung zum Server aufweisen, sondern im Wesentlichen demonstrieren, wie die Produktpflege im Web funktionieren könnte. Da hab' ich dann an Sie gedacht…

Products

Name	Price in €
3Doodler 3D Printing Pen	29.99
Powerstation 5- E. Maximus Chargus	44.95
8-Bit Legendary Hero Heat-Change Mug	6.99

Perfect new product € 123 Add

Abb. 10.1 *Erster Prototyp des Produktmanagers (mit Hinzufügen-Feature)*

Nachdem Sie mit Björn die genauen Anforderungen detailliert besprochen haben, können Sie ein HTML-Gerüst mit Produkt-Beispielen für das Produktpflege-Interface liefern:

```
1 <h1>Products</h1>
2 <table id="products">
3   <thead>
4     <tr>
```

```
5         <th>Name</th>
6         <th>Price in €</th>
7       </tr>
8     </thead>
9     <tbody>
10      <tr>
11        <td>3Doodler 3D Printing Pen</td>
12        <td>29.99</td>
13      </tr>
14      <tr>
15        <td>Powerstation 5- E. Maximus Chargus</td>
16        <td>44.95</td>
17      </tr>
18      <tr>
19        <td>8-Bit Legendary Hero Heat-Change Mug</td>
20        <td>6.99</td>
21      </tr>
22    </tbody>
23  </table>
24  <p id="new_product">
25    <input type="text" placeholder="name" class="name"/>
26    € <input type="number" max="999" placeholder="price"
class="price"/>
27    <button id="add_product">Add</button>
28  </p>
```

Codebeispiel 10.1 *additional_files/10/examples/product_manager_1/product_manager.html*

Erst einmal sollte der *Add*-Button funktionieren. Dazu registrieren Sie — wie üblich — eine Funktion auf das Click-Event:

```
"use strict";
{
  const init = () => {
    $("#add_product").on("click", addProduct);
  };
  const addProduct = () => {
    ...
  };
  init();
}
```

10.1 HTML generieren lassen mit document.createElement

Was genau soll `addProduct` tun? Ein möglicher Ansatz wäre, mit `innerHTML` die Tabelle neu zu schreiben und dabei jeweils das neueste Produkt anzuhängen. Das könnten Sie bereits umsetzen.

Das hätte aber den gravierenden Nachteil, dass Sie jedes Mal die Tabelle neu ins HTML schreiben müssten, obwohl nur eine neue Zeile dazukommt. Ein weiterer Nachteil ist, dass Sie mit eher »weichen« Strings arbeiten statt mit einen DOM-

Teilbaum. Strings sind anfällig gegenüber Syntaxfehlern und Sie können keine Methoden des Node-Objekts anwenden, solange Sie den String noch nicht mittels `innerHTML` in die Seite »eingehängt« haben.

Besser wäre es, komplette HTML-***Teilbäume*** zu konstruieren und diese dann bei Bedarf in den Hauptbaum einzuhängen.

Das ist ohne Weiteres möglich!

Dazu sind ein paar neue Methoden nötig. Mit `document.createElement` können Sie ein *HTMLElement*-Objekt erzeugen, also hier zunächst mal ein `td`.

```
const addProduct = () => {
  const valueTd = document.createElement("td");
  ...
};
```

Diesem können Sie dann mittels `textContent` einen Text als Inhalt zuweisen, in diesem Fall den neuen Produktnamen aus dem Eingabefeld.

```
const addProduct = () => {
  const valueTd = document.createElement("td");
  valueTd.textContent = $("#new_product .name").value;
  ...
};
```

Lassen Sie sich das erzeugte Objekt einfach mal mit `console.log(valueTd);` ausgeben:

```
const addProduct = () => {
  const valueTd = document.createElement("td");
  valueTd.textContent = $("#new_product .name").value;
  console.log(valueTd);
};
```

Geben Sie als Produktname testweise z.B. *Mein Neues Produkt* und *100€* ein. Sie erhalten:

```
<td>Mein Neues Produkt</td>
```

Es handelt sich bei `valueTd` aber nicht etwa um den String `<td>Mein Neues Produkt</td>`, sondern tatsächlich um ein `HTMLTableCellElement`, eine Spezialisierung von `HTMLElement` bzw. von `Node`. Letzteres bedeutet, Sie könnten z.B. die `classList` manipulieren oder Event-Handler dranhängen, was mit einem String so nicht machbar wäre.

10.2 Endlich angefügt: appendChild

Um die Tabellenzeile zu vervollständigen, müssen Sie noch das `td` mit dem Preis bauen und anschließend beide Elemente an ein neues `tr` anhängen. Zum Anhängen verwenden Sie die Methode `appendChild` des Node-Objekts:

```
const addProduct = () => {
  const valueTd = document.createElement("td");
  valueTd.textContent = newProductName();
  const priceTd = document.createElement("td");
  priceTd.textContent = newProductPrice();
  const productTr = document.createElement("tr");
  productTr.appendChild(valueTd);
  productTr.appendChild(priceTd);
  console.log("productTr", productTr);
}
```

Das `console.log` am Ende der Funktion zeigt Ihnen nun das komplette HTML-Fragment der Tabellenzeile:

```
<tr>
  <td>Mein Neues Produkt</td>
  <td>100</td>
</tr>
```

Sobald Sie dieses Fragment an der richtigen Stelle (dem `tbody` der Tabelle) in den HTML-Hauptbaum einhängen, erscheint die neue Zeile.

```
...
$("#products > tbody").appendChild(productTr);
```

Hier nochmal zur Übersicht der komplette Code:

```
"use strict";

{
  const init = () => {
    $("#add_product").on("click", addProduct);
  };

  const addProduct = () => {
    const valueTd = document.createElement("td");
    valueTd.textContent =  $("#new_product .name").value;

    const priceTd = document.createElement("td");
    priceTd.textContent = $("#new_product .price").value;

    const productTr = document.createElement("tr");
    productTr.appendChild(valueTd);
    productTr.appendChild(priceTd);
```

```
    $("#products > tbody").appendChild(productTr);
  };

  init();
}
```

Codebeispiel 10.2 *additional_files/10/examples/product_manager_1/product_manager.js*

Natürlich ist der Code nicht gerade ein Kunstwerk. Das erwartet auch niemand — aber zumindest etwas mehr Lesbarkeit wäre gut. Da Sie keinen Ärger mit Carl dem Consultant wollen, ist noch ein wenig Refactoring angesagt.

10.3 Mal wieder etwas Refactoring

Das Kernproblem besteht darin, dass die Funktion `addProduct` viele zu viele Verantwortlichkeiten hat. Sie

- nimmt die Produktdaten aus dem Formular entgegen
- fügt das Produkt grundsätzlich hinzu
- kümmert sich um Low-level-Belange, wie das Erzeugen der benötigten `tr`- und `td`-Elemente.

Genau diese Verantwortlichkeiten sollten Sie trennen (***Separation of Concerns***), anderenfalls enstehen unangenehme Redundanzen. Der Zugriff auf `createElement`, `appendChild` und `textContent` passiert möglicherweise öfter als nötig. Das kann ein erster Hinweis darauf sein, wo es sich lohnt, Funktionalität auszulagern.

Beginnen wir mit den Low-level-Belangen. In folgendem Code zeichnet sich ein Muster ab:

```
const valueTd = document.createElement("td");
valueTd.textContent = $("#new_product .name").value;
const priceTd = document.createElement("td");
priceTd.textContent = $("#new_product .price").value;
```

Der Code erzeugt zweimal ein `td` und weist diesem einen Textinhalt zu. Wie wäre es mit einer Methode namens — Achtung … — `td`, die genau das tut?

```
const td = text => {
  const tdNode = document.createElement("td");
  tdNode.textContent = text;
  return tdNode;
};
```

Damit beleibt von `addProduct` übrig:

```
const addProduct = () => {
  const productTr = document.createElement("tr");
  productTr.appendChild(td($("#new_product .name").value));
  productTr.appendChild(td($("#new_product .price").value));
  $("#products > tbody").appendChild(productTr);
};
```

Es ist naheliegend, mit dem `tr`-Element genauso zu verfahren. Der Unterschied ist, dass das `tr` statt eines Textes mehrere `td`-Elemente (als Array) entgegennehmen kann.

```
const tr = tds => {
  const trNode = document.createElement("tr");
  tds.forEach(td => trNode.appendChild(td));
  return trNode;
};
```

Ein erneuter Blick auf `addProduct` zeigt:

```
const addProduct = () =>
  $("#products > tbody").appendChild(
    tr([
      td($("#new_product .name").value),
      td($("#new_product .price").value)
    )];
```

Als Letztes gilt es noch, das Auslesen der Produktdaten vom Hinzufügen des Produkts zu trennen. Es kann gut sein, dass Sie später mal Produkte hinzufügen möchten, deren Daten nicht aus den Formularfeldern kommen (Achtung, Spoiler):

```
const init = () => {
  $("#add_product").on("click", addProductFromInput);
};
const addProductFromInput = () =>
  addProduct(
    $("#new_product .name").value,
    $("#new_product .price").value
  );
const addProduct = (name, price) =>
  $("#products > tbody").appendChild(
    tr([
      td(name), td(price)
    ])
  );
```

...nach etwa 2 Minuten und 33½ Sekunden Refactoring kommen wir also zu folgendem Gesamtergebnis:

```
"use strict";

{
  const init = () => {
    $("#add_product").on("click", addProductFromInput);
  };

  const addProductFromInput = () =>
    addProduct(
      $("#new_product .name").value,
      $("#new_product .price").value
    );

  const addProduct = (name, price) =>
    $("#products > tbody").appendChild(
      tr([
        td(name), td(price)
      ])
    );

  const td = text => {
    const tdNode = document.createElement("td");
    tdNode.textContent = text;
    return tdNode;
  };

  const tr = tds => {
    const trNode = document.createElement("tr");
    tds.forEach(td => trNode.appendChild(td));
    return trNode;
  };

  init();
}
```

Codebeispiel 10.3 *additional_files/10/examples/product_manager_2_refact/product_manager.js*

Übung 21: 01001000010011110101010 0 — Teil 7

Verändern Sie die Produktseite der Binär-Tasse mit Hilfe von JavaScript. Sie können wieder die JS-Konsole verwenden.

1. Fügen Sie in der Liste der `Product Specifications` mit Hilfe von JavaScript folgenden Listenpunkt hinzu:

   Capacity: 11 oz.

2. Erweitern Sie die Select-Box neben dem buy-Button mit Hilfe von JavaScript um den Eintrag: *5 items.*

10.4 Und noch ein wenig JSON

Könnten Sie die drei Produkte aus dem HTML herausnehmen? Mir wäre es lieber, wenn wir sie über eine externe Datei pflegen könnten. Ich weiß, ich weiß … es ist ein Prototyp. Aber um den sinnvoll zu testen, ist es auch sinnvoll, wenn wir die Produkte schnell ändern könnten, ohne das HTML anzupassen.

Für unsere DB haben wir einen JSON-Export — so könnten wir bei Bedarf einfach mal schnell mit ein paar echten Produkten testen. Können wir die Produkte in der Zusatzdatei als JSON (***JavaScript Object Notation***) speichern?

Natürlich lässt sich das einrichten! Die Datei für die drei Produkte sieht folgendermaßen aus:

```
1 "use strict";
2
3 const PRODUCTS = [
4   {name: "3Doodler 3D Printing Pen", price: 29.99},
5   {name: "Powerstation 5- E. Maximus Chargus", price: 44.95},
6   {name: "8-Bit Legendary Hero Heat-Change Mug", price: 6.99}
7 ];
```

Codebeispiel 10.4 *additional_files/10/examples/product_list.js*

In der Konstante `PRODUCTS` sind die drei Produkte in einem Array gespeichert. Jedes der Array-Elemente ist ein Objekt und hat die Attribute `name` und `price` mit den entsprechenden Werten.

Wenn Sie diese Daten bequem verwenden möchten, sollten Sie zunächst ein weiteres Refactoring des Produktmanager-Codes durchführen: Statt der Funktion `addProduct` jeweils `name` und `price` getrennt zu übergeben, wäre es hilfreich, diese zu einem Objekt zusammenzufassen, wie Sie sie im `PRODUCTS`-Array vorfinden — z.B. `{name: "3Doodler 3D Printing Pen", price: 29.99}`.

Ändern Sie dazu zunächst die Funktion `addProduct`. Innerhalb der Funktion können Sie auf die ursprünglichen Werte `name` und `price` als Attribute `product.name` und `product.price` des übergebenen Objekts `product` zugreifen.

```
const addProduct = product =>
  $("#products > tbody").appendChild(
    tr([
      td(product.name), td(product.price)
    ])
  );
...
```

Passen Sie den Aufruf von `addProduct` in der Funktion `addProductFromInput` an. Damit der Code wieder funktioniert, müssen Sie `addProduct` nun ein Objekt übergeben. Dieses Objekt erstellen Sie direkt beim Aufruf aus den Formularwerten für `name` und `price`.

```
...
const addProductFromInput = () =>
  addProduct({
    name: $("#new_product .name").value,
    price: $("#new_product .price").value
  });
```

`addProduct` arbeitet nun mit ganzen Objekten statt nur mit einzelnen Werten. Das macht es leicht, die Objekte aus der `PRODUCTS`-Konstante beim Laden hinzuzufügen. Schreiben Sie dazu eine neue Funktion `addExistingProducts`, die mit `forEach` alle Produkte hinzufügt.

```
const addExistingProducts = () => PRODUCTS.forEach(addProduct);
```

Rufen Sie sie in `init` auf.

```
const init = () => {
  addExistingProducts(PRODUCTS);
  $("#add_product").on("click", addProductFromInput);
};
```

Hurra, die Produkte werden importiert. Wie üblich, ist hier nochmal der Code im Ganzen:

```
"use strict";

{
  const init = () => {
    addExistingProducts(PRODUCTS);
    $("#add_product").on("click", addProductFromInput);
  };

  const addExistingProducts = () => PRODUCTS.forEach(addProduct);

  const addProductFromInput = () =>
    addProduct({
      name: $("#new_product .name").value,
      price: $("#new_product .price").value
    });
```

```
  const addProduct = product =>
    $("#products > tbody").appendChild(
      tr([
        td(product.name), td(product.price)
      ])
    );

  const td = text => {
    const tdNode = document.createElement("td");
    tdNode.textContent = text;
    return tdNode;
  };

  const tr = tds => {
    const trNode = document.createElement("tr");
    tds.forEach(td => trNode.appendChild(td));
    return trNode;
  };

  init();
}
```

Codebeispiel 10.5 *additional_files/10/examples/product_manager_3_json/product_manager.js*

Ha — ich hab' gerade eine riiieesen-Datei mit Produkten im JSON-Format ausprobiert. Funktioniert perfekt!

10.5 Vergleich verschiedener DOM-Creation-Methoden

Sie haben in diesem Kurs verschiedene Möglichkeiten zur Erzeugung von DOM-Elementen kennengelernt. Tabelle 10.1 zeigt die verschiedenen Möglichkeiten noch mal im Vergleich. Dazu gehen wir von folgender einfachen HTML-Struktur aus:

```
...
<body>
    <p>World</p>
</body>
...
```

JavaScript-Anweisung	HTML-Struktur nach der Ausführung

`document.body`
`  .innerHTML = "<p>Hello</p>";`
	`<body>`
	`  <p>Hello</p>`
	`<body>`

`document.body` `  .textContent = "Hello";`	`<body>Hello</body>`

`document.body`
`  .appendChild($(p).cloneNode());`
	`<body>`
	`  <p>World</p>`
	`  <p>World</p>`
	`<body>`

`document.body` `  .appendChild(` `  document.createElement("br")` `  );`	`<body>` `  <p>World</p>` `   ` `<body>`

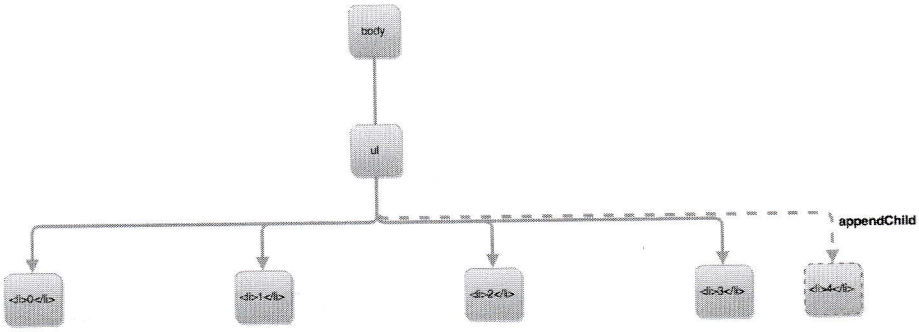
Tabelle 10.1 *DOM-Creation-Methoden im Vergleich*

10.6 Zusammenfassung

Methode	Beschreibung
appendChild	Hängt einen übergebenen Knoten (z. B. ein Element) als letztes Kind an den Elternknoten an, auf dem die Methode aufgerufen wurde. Falls der angehängte Knoten bereits Teil des Dokuments war, wird er von seiner ursprünglichen Position entfernt.
document.createElement	Erzeugt ein Element des angegebenen Typs.
cloneNode	Kopiert einen Knoten und gibt die Kopie zurück. `cloneNode(true)` kopiert auch alle Kindknoten mit.

Tabelle 10.2 *Methoden dieser Lektion*

Beispiel zu appendChild mit createElement



HTML-Struktur (vorher):

```
...
<body>
  <ul>
    <li>0</li>
    <li>1</li>
    <li>2</li>
    <li>3</li>
  </ul>
</body>
```

JavaScript in der Konsole:

```
const li = document.createElement("li");
li.textContent = "4";
$("ul").appendChild(li);
```

HTML-Struktur (nachher):

```
...
<body>
  <ul>
    <li>0</li>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
  </ul>
</body>
```

Beispiel zu appendChild *mit* cloneNode

HTML-Struktur (vorher):

```
...
<body>
  <ul>
    <li>0</li>
    <li>1</li>
    <li>2</li>
    <li>3</li>
  </ul>
</body>
```

JavaScript in der Konsole:

```
const li = $("ul li:first").clone(true);
$("ul").appendChild(li);
```

HTML-Struktur (nachher):

```
...
<body>
  <ul>
    <li>0</li>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
    <li>0</li>
  </ul>
</body>
```

Beispiel zu appendChild *ohne* cloneNode

HTML-Struktur (vorher):

```
...
<body>
  <ul>
    <li>0</li>
    <li>1</li>
    <li>2</li>
    <li>3</li>
  </ul>
</body>
```

JavaScript in der Konsole:

```
const li = $("ul li:first");
$("ul").appendChild(li);
```

HTML-Struktur (nachher):

```
...
<body>
  <ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
    <li>0</li>
  </ul>
</body>
```

Ab in die Tonne mit remove 11

Ich hab' mal testweise massig Produkte per Formular hinzugefügt. Davon würde ich aber gerne ein paar wieder löschen. Können Sie einen Entfernen-Button hinzufügen?

Products

Name	Price in €	Actions
3Doodler 3D Printing Pen	29.99	X
Powerstation 5- E. Maximus Chargus	44.95	X
8-Bit Legendary Hero Heat-Change Mug	6.99	X

Abb. 11.1 *Produktmanager: Entfernen-Feature*

Damit jedes Produkt über einen eigenen Entfernen-Button verfügt, ist eine weitere Spalte in der Tabelle hilfreich. Weil später neben dem Löschbutton noch weitere Aktionen folgen, fügen Sie der Tabelle im HTML eine *Actions*-Spalte hinzu.

```
<table id="products">
    <tr>
      <th>Name</th>
      <th>Price in €</th>
      <th>Actions</th>
    </tr>
</table>
```

Erweitern Sie die Funktion `addProduct`. Jede Produkt-Zeile benötigt einen weiteren TableData-Tag (`td`) mit dem Entfernen-Button (RemoveButton).

```
1 const addProduct = product =>
2   $("#products > tbody").appendChild(
3     tr([
4       td(product.name), td(product.price), tdWithRemoveButton()
5     ])
6   );
```

Es fehlt noch die Funktion `tdWithRemoveButton`, die das entsprechende HTML-Fragment anlegt.

```
const tdWithRemoveButton = () => {
  const td = document.createElement("td");
  td.appendChild(removeButton());
  return td;
};
const removeButton = () => {
  const button = document.createElement("button");
  button.innerText = "x";
  button.classList.add("remove_product");
  return button;
};
```

Die Aufteilung auf zwei Funktionen verbessert die Lesbarkeit. Der Remove-Button zeigt ein `x` als Entfernen-Symbol und erhält eine passende CSS-Klasse `remove_product`. Jetzt zur Funktionalität:

Registrieren Sie eine Funktion `removeProduct` auf das `click`-Event.

```
...
onst removeButton = () => {
   button.on("click", removeProduct);
   return button;
;
```

`removeProduct` muss die ganze Tabellen-Zeile (`tr`) mit dem Produkt entfernen. Dazu müssen Sie sie erst finden. Wenn Sie vom Button aus (hier `event.target`) zwei Ebenen in der Baumstruktur nach oben wechseln, haben Sie die Zeile. Das Nach-oben-Wechseln lässt sich mit dem Attribut `parentNode` erreichen — Sie suchen den jeweiligen Elternknoten.

```
event.target.parentNode.parentNode
```

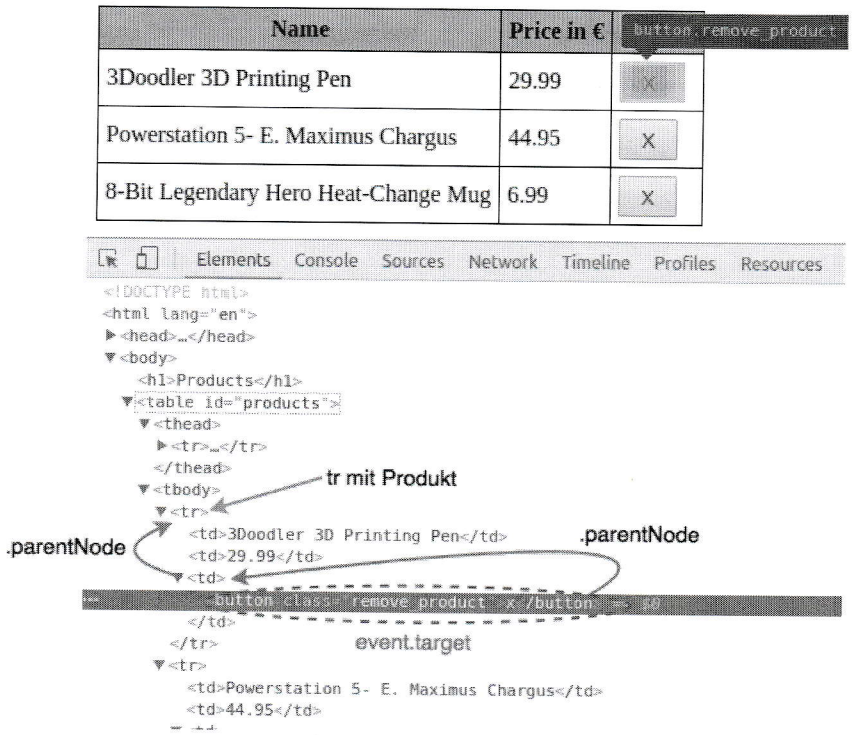


Abb. 11.2 *Productmanager: Vom Button zur Produkt-Zeile*

Das Attribut `parentNode` gehört zu einer Gruppe von Attributen, die es Ihnen erlauben, sich innerhalb des DOM-Baums von einem Knoten (Node) zum anderen zu bewegen. Der Bereich, in dem diese Attribute eingeordnet sind, heißt ***DOM-Traversal*** — damit beschäftigen wir uns gleich noch näher.

Entfernen Sie die gefundene Zeile (`tr`) mit der `remove`-Methode des zur Zeile gehörigen Node-Objekts — und sie ist weg!

```
const removeProduct = event =>
  event.target.parentNode.parentNode.remove();
```

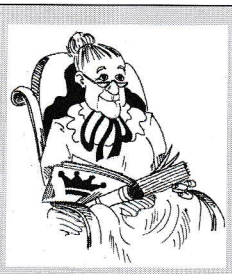
Nach einem Extract-Refactoring erhalten Sie folgenden, etwas besser lesbaren Code:

```
const removeProduct = event =>
  productRowForAction(event.target).remove();
const productRowForAction = button => button.parentNode.parentNode;
```

Und hier ist nochmal der komplette Code am Stück:

```
1 "use strict";
2
3 {
4   const init = () => {
5     addExistingProducts(PRODUCTS);
6     $("#add_product").on("click", addProductFromInput);
7   };
8
9   const addExistingProducts = () => PRODUCTS.forEach(addProduct);
10
11  const addProductFromInput = () =>
12    addProduct({
13      name: $("#new_product .name").value,
14      price: $("#new_product .price").value
15    });
16
17  const addProduct = product =>
18    $("#products > tbody").appendChild(
19      tr([
20        td(product.name), td(product.price), tdWithRemoveButton()
21      ])
22    );
23
24  const tdWithRemoveButton = () => {
25    const td = document.createElement("td");
26    td.appendChild(removeButton());
27    return td;
28  };
29
30  const removeButton = () => {
31    const button = document.createElement("button");
32    button.textContent = "x";
33    button.classList.add("remove_product");
34    button.on("click", removeProduct);
35    return button;
36  };
37
38  const removeProduct = event =>
39    productRowForAction(event.target).remove();
40
41  const productRowForAction = button =>
42    button.parentNode.parentNode;
43
44  const td = text => {
45    const tdNode = document.createElement("td");
46    tdNode.textContent = text;
47    return tdNode;
48  };
49
50  const tr = tds => {
51    const trNode = document.createElement("tr");
52    tds.forEach(td => trNode.appendChild(td));
53    return trNode;
54  };
55
56  init();
57 }
```

Codebeispiel 11.1 *additional_files/11/examples/product_manager_4_remove/product_manager.js*



»Also früher, da mussten wir noch ...« remove vs. removeChild

Die `remove`-Methode des Element-Objektes ist eine eher neue Entwicklung. Im Internet Explorer (einschließlich IE11) steht sie nicht zur Verfügung. Nur moderne Browser wie Chrome, Firefox und Edge haben sie direkt an Bord.

Das ist aber nicht weiter tragisch. In älteren Browsern gab es bereits die ähnliche Methode `removeChild`. Damit lässt sich sehr leicht der folgende Polyfill konstruieren:

```
if (!('remove' in Element.prototype)) {
    Element.prototype.remove = function() {
        if (this.parentNode) {
            this.parentNode.removeChild(this);
        }
    };
}
```

Codebeispiel 11.2 *Quelle: https://developer.mozilla.org/en-US/docs/Web/API/ChildNode/remove*

Fügen Sie den Polyfill einfach in Ihre *dom_helper*-Library ein, falls Sie ältere Browser unterstützen möchten.

11.1 Klassifizierung

Übrigens ist es recht hilfreich, DOM-Eigenschaften nach Einsatzzweck zu klassifizieren. Eine Klassifizierung erleichtert den Überblick über die Vielzahl an Eigenschaften und ist gleichzeitig hilfreich, wenn Sie etwas Bestimmtes suchen.

Typischerweise lassen sich die DOM-Attribute und -Methoden vor allem in diese drei Bereiche untergliedern:

- DOM-Selection
- DOM-Manipulation
- DOM-Traversal

Der Bereich ***DOM-Selection*** beinhaltet vor allem Methoden wie `document.querySelectorAll`, die Ihnen ermöglichen, DOM-Elemente aufzustöbern. Mit Hilfe von DOM-Manipulation-Methoden wie `classList.add` oder Eigenschaften wie `.innerHTML` können Sie diese dann verändern.

Wir verwenden die Klassifizierung übrigens auch hinten im Referenz-Anhang, um Ihnen das Nachschlagen zu erleichtern.

11.2 Zusammenfassung

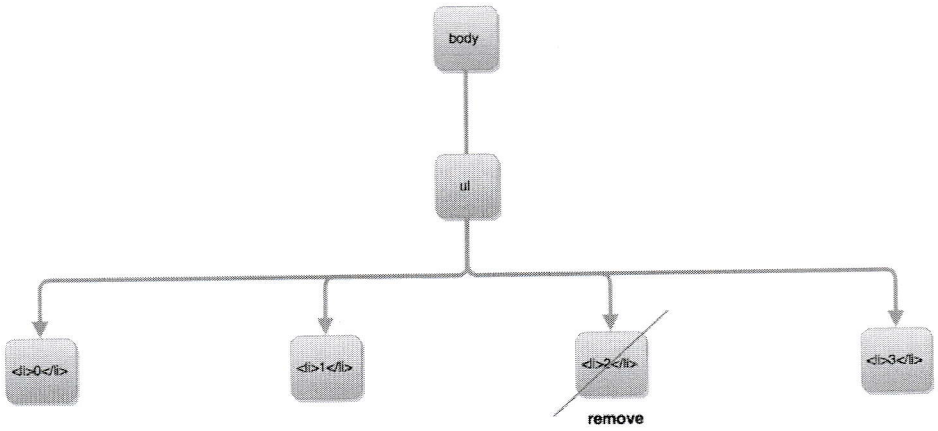
Methode	Beschreibung
remove	Entfernt einen Knoten aus seinem HTML-Baum. Der Knoten wird zurückgegeben und kann weiterverwendet werden.

Tabelle 11.1 *Methoden dieser Lektion*

Attribut	Beschreibung
parentNode	Enthält den Eltern-Knoten des aktuellen Elements.

Tabelle 11.2 *Attribute dieser Lektion*

Beispiel zu `remove`



HTML-Struktur (vorher):

```
...
<body>
  <ul>
    <li>0</li>
    <li>1</li>
    <li>2</li>
    <li>3</li>
  </ul>
</body>
```

JavaScript in der Konsole

```
$$("li")[2].remove();
```

HTML-Struktur (nachher):

```
...
<body>
  <ul>
    <li>0</li>
    <li>1</li>
    <li>3</li>
  </ul>
</body>
```

11.3 Übungen

Übung 22: 01001000010011110101010 0 — Teil 8

Verändern Sie erneut die Produktseite der Binär-Tasse mit Hilfe von JavaScript. Sie können auch hier wieder die JS-Konsole verwenden.

1. Entfernen Sie die Überschrift `Description` mit der `remove`-Methode.
2. Entfernen Sie mit der remove-Methode den Text `"01001000010011110101010 0"` aus der Hauptüberschrift.

12 Wenn Geschwister eine Reise tun: siblings & insertBefore

Als Letztes wäre es noch schön, wenn es neben dem Remove-Button noch 2 Pfeil-Buttons für »*hoch*« und »*runter*« gäbe, um das jeweilige Produkt zu verschieben. Dann könnten wir mit dem Prototyp erst mal experimentieren und entscheiden, ob wir diese Funktionalität tatsächlich so im Endprodukt haben möchten.

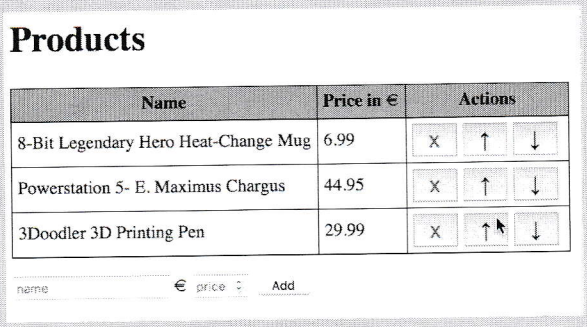


Abb. 12.1 *Produktmanager: Hoch&Runter-Feature*

Zum Ergänzen der beiden neuen Buttons erweitern Sie die Funktion `tdWithRemoveButton` und benennen sie am besten auch gleich um, da sie danach mehr als nur den Remove-Button erzeugt — in `tdWithActionButtons`.

```
1 const tdWithActionButtons = () => {
2     const td = document.createElement("td");
3     td.appendChild(removeButton());
4     td.appendChild(moveUpButton());
5     td.appendChild(moveDownButton());
6     return td;
7 };
```

Die Buttons erzeugen Sie nach dem bekannten Schema:

```
const moveUpButton = () => {
  const button = document.createElement("button");
  button.innerText = "↑";
  button.classList.add("move_product_up");
  button.on("click", moveProductUp);
```

```
  return button;
};
const moveDownButton = () => {
  const button = document.createElement("button");
  button.innerText = "↓";
  button.classList.add("move_product_down");
  button.on("click", moveProductDown);
  return button;
};
```

Ja, wir wissen schon, was Sie sagen wollen: zuviel Redundanz durch böses Copy&Paste. Natürlich haben Sie recht. Aber darum kümmern wir uns später, da wir zuerst noch die Funktionen `moveProductUp` und `moveProductDown` implementieren müssen.

Wie sehen nun die eigentlichen Bewegungsfunktionen `moveProductUp` und `moveProductDown` aus? Auch hier müssen Sie sich zunächst die Produkt-Zeile schnappen — mit der existierenden Funktion `productRowForAction(event.target)`, die einfach wieder zwei parentNodes nach oben wechselt. Speichern Sie sie gleich in einer Variablen:

```
const currentProductRow = productRowForAction(event.target);
```

Jetzt können Sie diese Produktzeile an einer anderen Stelle wieder einfügen. Dazu benötigen Sie die Methode `insertBefore`. Sie kann ein Element vor einem anderen einfügen. Aufrufen müssen Sie die Methode immer auf dem Eltern-Element:

```
currentProductRow.parentNode.insertBefore(/* ... */)
```

Wo genau soll das Element eigentlich hin?

Zum Nach-oben-Bewegen müssen Sie es auf der gleichen Ebene, ein Element vor der aktuellen Produktuzeile, einfügen. Elemente der gleichen Ebene heißen Geschwister (engl. ***siblings***). Das Geschwister-Element vor dem aktuellen ist dann der `previousElementSibling` (vorheriges Geschwister-Element). In Code bedeutet das:

```
1 currentProductRow.parentNode.insertBefore(
2   currentProductRow,
3   currentProductRow.previousElementSibling);
```

Das sieht etwas kompliziert aus, ist es aber nicht. Lesen Sie die Anweisung als:

> *Ausgehend vom Parent (`currentProductRow.parentNode` — das ist hier der `tbody`) — füge die zu verschiebende Produktzeile (`currentProductRow`) **vor** dem*

vorhergehenden Geschwister-Element
(`currentProductRow.previousElementSibling`) *ein*.

Damit erhalten Sie folgende Funktion:

```
const moveProductUp = event => {
  const currentProductRow = productRowForAction(event.target);
  currentProductRow.parentNode.insertBefore(
    currentProductRow,
    currentProductRow.previousElementSibling);
};
```

Jetzt lässt sich das Produkt nach oben schieben — hurra! Probieren Sie es aus.

`moveProductDown` ist auch kein Kunststück mehr. Sie benötigen das Attribut `nextElementSibling`. Leider gibt es ein kleines Problem. Das nächste Geschwister-`tr` von *3Doodler*-`tr` ist die *Powerstation*. Wenn Sie den 3Doodle mit `insertBefore` vor der *Powerstation* einfügen (*before next* in Abb. 12.2), haben Sie nichts verändert!

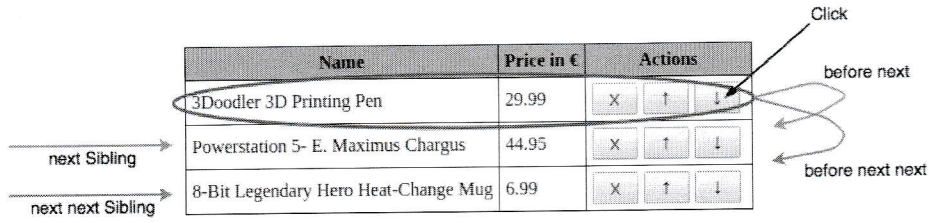


Abb. 12.2 *moveProductDown mit* `insertBefore` *auf* `.nextElementSibling.nextElementSibling`

Sie müssen den *3Doodler* stattdessen vor dem übernächsten `tr` (*before next next* in Abb. 12.2), der *8.Bit…Mug*, einfügen:

```
const moveProductDown = event => {
  const currentProductRow = productRowForAction(event.target);
  currentProductRow.parentNode.insertBefore(
    currentProductRow,
    currentProductRow.nextElementSibling.nextElementSibling);
};
```

Hier ist nochmal alles im Überblick:

```
"use strict";

{
  const init = () => {
    addExistingProducts(PRODUCTS);
    $("#add_product").on("click", addProductFromInput);
  };

  const addExistingProducts = () => PRODUCTS.forEach(addProduct);
```

```
10
11  const addProductFromInput = () =>
12    addProduct({
13      name: $("#new_product .name").value,
14      price: $("#new_product .price").value
15    });
16
17  const addProduct = product =>
18    $("#products > tbody").appendChild(
19      tr([
20        td(product.name), td(product.price), tdWithActionButtons()
21      ])
22    );
23
24  const tdWithActionButtons = () => {
25    const td = document.createElement("td");
26    td.appendChild(removeButton());
27    td.appendChild(moveUpButton());
28    td.appendChild(moveDownButton());
29    return td;
30  };
31
32  const removeButton = () => {
33    const button = document.createElement("button");
34    button.textContent = "x";
35    button.classList.add("remove_product");
36    button.on("click", removeProduct);
37    return button;
38  };
39
40  const moveUpButton = () => {
41    const button = document.createElement("button");
42    button.textContent = "↑";
43    button.classList.add("move_product_up");
44    button.on("click", moveProductUp);
45    return button;
46  };
47
48  const moveDownButton = () => {
49    const button = document.createElement("button");
50    button.textContent = "↓";
51    button.classList.add("move_product_down");
52    button.on("click", moveProductDown);
53    return button;
54  };
55
56  const removeProduct = event =>
57    productRowForAction(event.target).remove();
58
59  const moveProductUp = event => {
60    const currentProductRow = productRowForAction(event.target);
61    currentProductRow.parentNode.insertBefore(
62        currentProductRow,
63        currentProductRow.previousElementSibling);
64  };
65
66  const moveProductDown = event => {
67    const currentProductRow = productRowForAction(event.target);
68    currentProductRow.parentNode.insertBefore(
```

```
69          currentProductRow,
70          currentProductRow.nextElementSibling.nextElementSibling);
71    };
72
73    const productRowForAction = button =>
74      button.parentNode.parentNode;
75
76    const td = text => {
77      const tdNode = document.createElement("td");
78      tdNode.textContent = text;
79      return tdNode;
80    };
81
82    const tr = tds => {
83      const trNode = document.createElement("tr");
84      tds.forEach(td => trNode.appendChild(td));
85      return trNode;
86    };
87
88    init();
89  }
```

Codebeispiel 12.1 *additional_files/12/examples/product_manager_5_move/product_manager.js*

Ganz schön lang, nicht wahr?

12.1 Allerliebstes Refactoring

Zumindest die Redundanzen sollten Sie noch entfernen. Alle drei Button-Funktionen sind nach dem gleichen Schema aufgebaut:

```
const removeButton = () => {
  const button = document.createElement("button");
  button.textContent = "x";
  button.classList.add("remove_product");
  button.on("click", removeProduct);
  return button;
};
const moveUpButton = () => {
  const button = document.createElement("button");
  button.textContent = "↑";
  button.classList.add("move_product_up");
  button.on("click", moveProductUp);
  return button;
};
const moveDownButton = () => {
  const button = document.createElement("button");
  button.textContent = "↓";
  button.classList.add("move_product_down");
  button.on("click", moveProductDown);
  return button;
};
```

Definieren Sie eine Funktion `buildButton`, die die Unterschiede vereinheitlicht:

```
const buildButton = (symbol, cssClass, action) => {
  const button = document.createElement("button");
  button.textContent = symbol;
  button.classList.add(cssClass);
  button.on("click", action);
  return button;
};
```

Sie können alle drei Buttons mit der neuen Funktion erzeugen:

```
const removeButton = () =>
  buildButton("x", "remove_product", removeProduct);
const moveUpButton = () =>
  buildButton("↑", "move_product_up", moveProductUp);
const moveDownButton = () =>
  buildButton("↓", "move_product_down", moveProductDown);
```

Der Code ist nicht nur kürzer — Sie können auf einen Blick erkennen, wie sich die drei Buttons tatsächlich voneinander unterscheiden.

Auch die beiden `move`-Funktionen unterscheiden sich nur minimal. Der einzige Unterschied ist der Zugriff auf die Geschwister-Elemente, um die Richtung festzulegen. Ziehen Sie die Unterschiede heraus:

```
const moveProductUp = event => moveProduct(event, up);
const moveProductDown = event => moveProduct(event, down);
const moveProduct = (event, direction) => {
  const currentProductRow = productRowForAction(event.target);
  currentProductRow.parentNode.insertBefore(
      currentProductRow, direction(currentProductRow));
};
const down = el => el.nextElementSibling.nextElementSibling;
const up = el => el.previousElementSibling;
```

Insgesamt sieht der verbesserte Code nun so aus:

```
1  "use strict";
2
3  {
4    const init = () => {
5      addExistingProducts(PRODUCTS);
6      $("#add_product").on("click", addProductFromInput);
7    };
8
9    const addExistingProducts = () => PRODUCTS.forEach(addProduct);
10
11   const addProductFromInput = () =>
12     addProduct({
13       name: $("#new_product .name").value,
14       price: $("#new_product .price").value
15     });
16
```

```
17  const addProduct = product =>
18    $("#products > tbody").appendChild(
19      tr([
20        td(product.name), td(product.price), tdWithActionButtons()
21      ])
22    );
23
24  const tdWithActionButtons = () => {
25    const td = document.createElement("td");
26    td.appendChild(removeButton());
27    td.appendChild(moveUpButton());
28    td.appendChild(moveDownButton());
29    return td;
30  };
31
32  const removeButton = () =>
33    buildButton("x", "remove_product", removeProduct);
34  const moveUpButton = () =>
35    buildButton("↑", "move_product_up", moveProductUp);
36  const moveDownButton = () =>
37    buildButton("↓", "move_product_down", moveProductDown);
38
39  const removeProduct = event =>
40      productRowForAction(event.target).remove();
41  const moveProductUp = event => moveProduct(event, up);
42  const moveProductDown = event => moveProduct(event, down);
43
44  const moveProduct = (event, direction) => {
45    const currentProductRow = productRowForAction(event.target);
46    currentProductRow.parentNode.insertBefore(
47        currentProductRow, direction(currentProductRow));
48  };
49
50  const down = el => el.nextElementSibling.nextElementSibling;
51  const up = el => el.previousElementSibling;
52
53  const productRowForAction = button =>
54    button.parentNode.parentNode;
55
56  const buildButton = (symbol, cssClass, action) => {
57    const button = document.createElement("button");
58    button.textContent = symbol;
59    button.classList.add(cssClass);
60    button.on("click", action);
61    return button;
62  };
63
64  const td = text => {
65    const tdNode = document.createElement("td");
66    tdNode.textContent = text;
67    return tdNode;
68  };
69
70  const tr = tds => {
71    const trNode = document.createElement("tr");
72    tds.forEach(td => trNode.appendChild(td));
73    return trNode;
```

```
74   };
75
76   init();
77 }
```

Codebeispiel 12.2 *additional_files/12/examples/product_manager_6_move_refact/product_manager.js*

12.2 Uups — das geht noch besser!

Mir ist noch ein kleiner Glitch aufgefallen. Ich kann beim ersten Produkt auf den »Nach-oben«-Button drücken. So sieht das Ergebnis aus:

3Doodler 3D Printing Pen	29.99	X ↑ ↓
Name	**Price in €**	**Actions**
Powerstation 5- E. Maximus Chargus	44.95	X ↑ ↓
8-Bit Legendary Hero Heat-Change Mug	6.99	X ↑ ↓

Genauso wenig Sinn macht es, beim letzten Produkt auf den »Nach-unten«-Button zu drücken. Könnten Sie die entsprechenden Buttons nicht einfach deaktivieren?

Stimmt, eigentlich sollten diese Buttons gar nicht aktiv sein. Ein kleine Funktion schafft hier Abhilfe.

```
1 const disableNonFunctionalButtons = () =>
2   $$("#products > tbody > tr").forEach(tr => {
3     tr.querySelector(".move_product_up").disabled =
!tr.previousElementSibling;
4     tr.querySelector(".move_product_down").disabled =
!tr.nextElementSibling;
5   });
```

Das ist nun im Grunde nichts Neues. Das `forEach` durchläuft alle Tabellenzeilen (`tr`) und schaltet die Rauf-/runter-Buttons in den richtigen Zustand. Der einzige Trick ist es, herauszufinden, was denn der richtige Zustand ist. Hat eine Zeile *keinen Vorgänger* (`!tr.previousElementSibling`), so muss sie die oberste Zeile sein. Dementsprechend ist der Button für »Nach oben« (`tr.querySelector(".move_product_up")`) in dieser Zeile zu deaktivieren (`.disabled = true`). Hat die Zeile jedoch einen Vorgänger, so gibt `!tr.previousElementSibling` den Wert `false` zurück. Sie setzen also das Attribut `disabled` auf `false`. Der Button bleibt (oder wird) aktiv.

Der »Nach-unten«-Button lässt sich genauso deaktivieren bzw. aktivieren.

Jetzt muss die Funktion nur noch aufgerufen werden. Wann ist der beste Zeitpunkt?

Idealerweise immer, nachdem der Anwender einen der Buttons betätigt hat.

Aufruf in der Tabelle

Für die Buttons `↑`, `↓` und `x` hängen Sie die Funktion `disableNonFunctionalButtons` einfach zusätzlich ans jeweilige Click-Event in der `buildButton`-Funktion. Das zweite `on` in Zeile 5 registriert `disableNonFunctionalButtons` als zweiten Event-Handler auf dem `click`-Event (***Chaining***). Beide Funktionen werden beim Auftreten des Events ausgeführt.

```
1 const buildButton = (symbol, cssClass, action) => {
2   const button = document.createElement("button");
3   button.innerText = symbol;
4   button.classList.add(cssClass);
5   button.on("click", action).on("click",
disableNonFunctionalButtons);
6   return button;
7 };
```

Jeder Klick auf einen von `buildButton` erzeugten Button ruft `disableNonFunctionalButtons` auf. Somit kann der Anwender einen beliebigen Button eines beliebigen Produktes betätigen und es bleibt sichergestellt, dass alle Buttons in den richtigen Zustand wechseln.

Aufruf nach dem Hinzufügen neuer Produkte

Um auch einen korrekten Zustand nach dem Hinzufügen neuer Produkte zu haben, sollten Sie die Funktion noch am Ende von `addProduct` einfügen:

```
1 const addProduct = product => {
2   $("#products > tbody").appendChild(
3     tr([
4       td(product.name), td(product.price), tdWithActionButtons()
5     ])
6   );
7   disableNonFunctionalButtons();
8 };
```

Wie immer der komplette Code zur Übersicht:

```
1 "use strict";
2
3 {
4   const init = () => {
5     addExistingProducts(PRODUCTS);
6     $("#add_product").on("click", addProductFromInput);
7   };
```

```
8
9   const addExistingProducts = () => PRODUCTS.forEach(addProduct);
10
11  const addProductFromInput = () =>
12    addProduct({
13      name: $("#new_product .name").value,
14      price: $("#new_product .price").value
15    });
16
17  const addProduct = product => {
18    $("#products > tbody").appendChild(
19      tr([
20        td(product.name), td(product.price), tdWithActionButtons()
21      ])
22    );
23    disableNonFunctionalButtons();
24  };
25
26  const tdWithActionButtons = () => {
27    const td = document.createElement("td");
28    td.appendChild(removeButton());
29    td.appendChild(moveUpButton());
30    td.appendChild(moveDownButton());
31    return td;
32  };
33
34  const removeButton = () =>
35    buildButton("x", "remove_product", removeProduct);
36  const moveUpButton = () =>
37    buildButton("↑", "move_product_up", moveProductUp);
38  const moveDownButton = () =>
39    buildButton("↓", "move_product_down", moveProductDown);
40
41  const removeProduct = event =>
42    productRowForAction(event.target).remove();
43  const moveProductUp = event => moveProduct(event, up);
44  const moveProductDown = event => moveProduct(event, down);
45
46  const moveProduct = (event, direction) => {
47    const currentProductRow = productRowForAction(event.target);
48    currentProductRow.parentNode.insertBefore(
49        currentProductRow, direction(currentProductRow));
50  };
51
52  const down = el => el.nextElementSibling.nextElementSibling;
53  const up = el => el.previousElementSibling;
54
55  const productRowForAction = button =>
56    button.parentNode.parentNode;
57
58  const buildButton = (symbol, cssClass, action) => {
59    const button = document.createElement("button");
60    button.textContent = symbol;
61    button.classList.add(cssClass);
62    button.on("click", action).on("click",
disableNonFunctionalButtons);
63    return button;
64  };
65
```

```
66   const disableNonFunctionalButtons = () =>
67     $$("#products > tbody > tr").forEach(tr => {
68       tr.querySelector(".move_product_up").disabled =
!tr.previousElementSibling;
69       tr.querySelector(".move_product_down").disabled =
!tr.nextElementSibling;
70     });
71
72   const td = text => {
73     const tdNode = document.createElement("td");
74     tdNode.textContent = text;
75     return tdNode;
76   };
77
78   const tr = tds => {
79     const trNode = document.createElement("tr");
80     tds.forEach(td => trNode.appendChild(td));
81     return trNode;
82   };
83
84   init();
85 }
```

Codebeispiel 12.3 *additional_files/12/examples/product_manager_7_move_disabled_buttons/product_manager.js*

12.3 Zusammenfassung

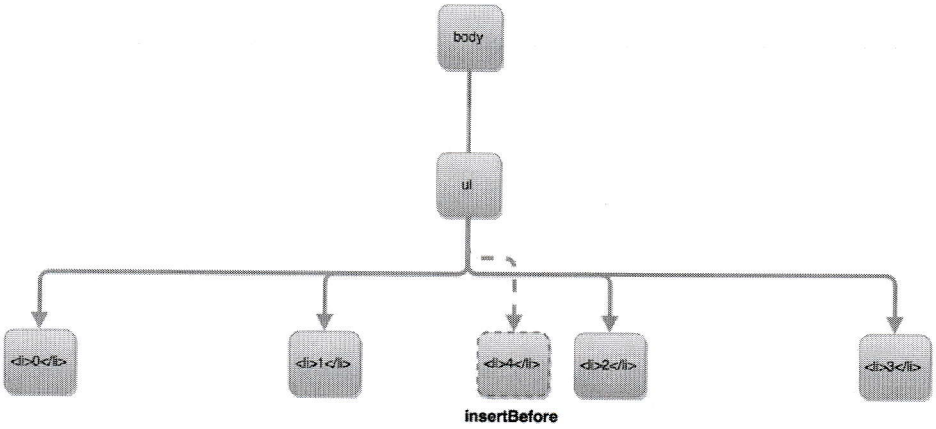
Methode	Beschreibung
insertBefore	Fügt einen übergebenen Knoten (erstes Argument) vor einem anderen Knoten (zweites Argument) ein. Ausgegangen wird dabei von dem Elternknoten, auf dem die Methode aufgerufen wurde.

Tabelle 12.1 *Methoden dieser Lektion*

Attribut	Beschreibung
previousElementSibling	Enthält das vorhergehende Geschwisterelement
nextElementSibling	Enthält das nachfolgende Geschwisterelement

Tabelle 12.2 *Attribute dieser Lektion*

12.3.1 Beispiel zu `insertBefore`



HTML-Struktur (vorher):

```
...
<body>
  <ul>
    <li>0</li>
    <li>1</li>
    <li>2</li>
    <li>3</li>
  </ul>
</body>
```

JavaScript in der Konsole

```
const li = document.create("li");
li.textContent = "4";
$("ul").insertBefore($$("ul > li")[2], li);
```

HTML-Struktur (nachher):

```
...
<body>
  <ul>
    <li>0</li>
    <li>1</li>
    <li>4</li>
    <li>2</li>
    <li>3</li>
  </ul>
</body>
```

12.3.2 DOM-Traversal

Um sich innerhalb des DOM von einem Element zum anderen zu bewegen (***traversal***), sind verschiedene Attribute nötig. Diese Übersicht zeigt die wichtigsten Attribute anhand von Codebeispiel 12.4.

Alle in Tabelle 12.3 bzw. Abb. 12.3 aufgelisteten Attribute sind *read-only*. Sie können zwar anhand der Attribute im DOM navigieren, aber keine Veränderungen an der Baumstruktur durchführen.

Attribut	Beschreibung
parentNode	der Elternknoten
children	alle Kindelemente
childElementCount	Anzahl der Kindelemente
firstElementChild	das erste Kindelelement
lastElementChild	das letzte Kindelelement
previousElementSibling	das vorhergehende Geschwisterelement
nextElementSibling	das nächste Geschwisterelement

Tabelle 12.3 *Diese Tabelle zeigt die wichtigsten Möglichkeiten zum DOM-**Traversal** (Bewegung innerhalb des DOMs)*

Beispiele

```
...
<body>
  <ul>
    <li>0</li>
    <li>1</li>
    <li>2</li>
    <li>3</li>
  </ul>
</body>
```

Codebeispiel 12.4 *HTML-Beispiel einer einfachen unordered list*

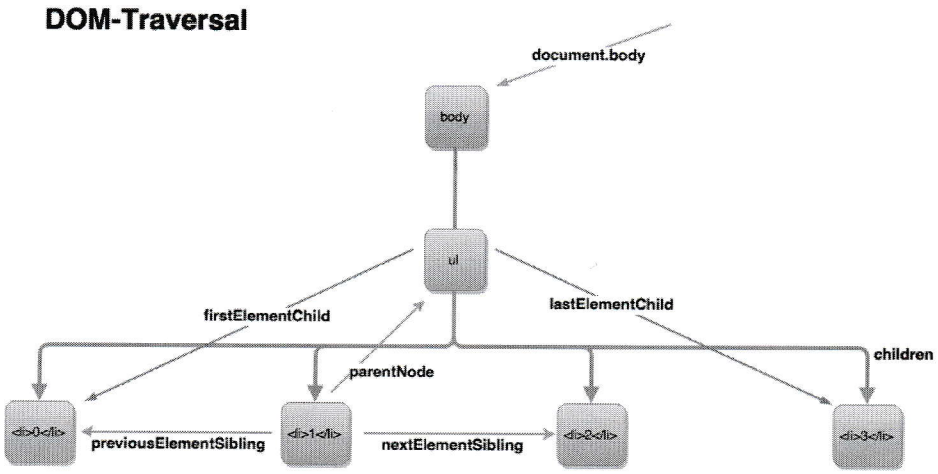
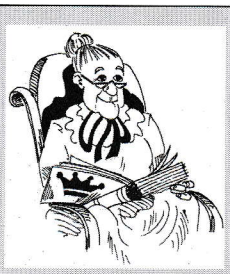


Abb. 12.3 *DOM-Traversal-Attributes zu Codebeispiel 12.4*

```
document.body                    // => <body>...</body>
document.body.children           // => [<ul>...</ul>]
document.body.childElementCount  // => 0
$('ul')                     // => <ul>...</ul>
$('ul').parentNode          // => <body>...</body>
$('ul').firstElementChild   // => <li>0</li>
$('ul').lastElementChild    // => <li>3</li>
$('ul').childElementCount   // => 4
$('ul').children // => [<li>0</li>, ..., <li>3</li>]
$$('li')[1]                          // => <li>1</li>
$$('li')[1].parentNode               // => <ul>...</ul>
$$('li')[1].previousElementSibling   // => <li>0</li>
$$('li')[1].nextElementSibling       // => <li>2</li>
```



Literaturhinweis

Die in den letzten Lektionen beschriebenen Techniken, um Elemente im DOM zu manipulieren, wurden vor allem durch das Buch *DOM Scripting* (Keith 2005) von *Jeremy Keith* bekannt. Es erschien 2005 und ritt damit auf dem Höhepunkt der Ajax-Hype-Welle. Es demonstrierte sehr eindrucksvoll, wie sich mit sauberem und standardkonformem Code moderne Anwendungen entwickeln lassen — ein Novum zur damaligen Zeit.

12.4 Übungen

Übung 23: 01001000010011110101010 0 — Teil 9

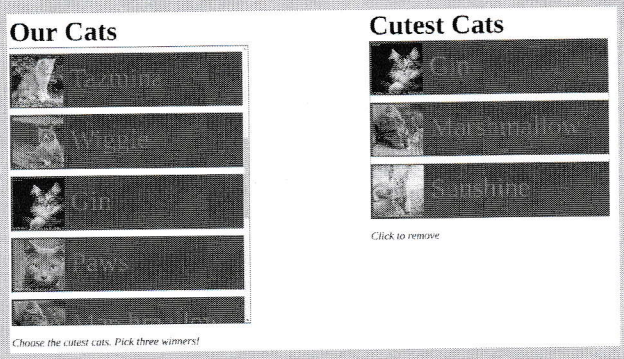
Verändern Sie noch mal die Produktseite der Binär-Tasse mit Hilfe von JavaScript.

Fügen Sie noch einmal den folgenden Listenpunkt zur Liste der `Product Specifications` hinzu:

Capacity: 11 oz.

Fügen Sie den Listenpunkt aber dieses Mal nicht am Ende der Liste ein, sondern vor dem Eintrag *Materials: Ceramic*.

Übung 24: Cutycat



Der Katzenfreundeverein *Cutycat* hat einen Auftrag für Sie. Der Verein möchte auf seiner Website einen Wettbewerb ausrichten. Die Seite zeigt eine Reihe von Katzen, aus denen ein Besucher jeweils die drei niedlichsten auswählen soll.

1. Verwenden Sie das HTML und CSS aus dem Begleitmaterial. Sobald ein Besucher auf eine der Katzen klickt, soll diese auch in der rechten Box (*Cutest Cats*) erscheinen. Sie darf aber aus der Liste der Katzen (*Our Cats*) nicht verschwinden!

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8" />
```

```
 6   <title>Cutycat</title>
 7   <script src="../../../lib/dom_helper.js"
defer="defer"></script>
 8   <script src="cutycat.js" defer="defer"></script>
 9   <link rel="stylesheet"  type="text/css"
href="cutycat.css" />
10 </head>
11
12 <body>
13
14   <section id="candidates_section">
15     <h1>Our Cats</h1>
16     <ul id="candidates" class="kittenlist">
17       <li><img src="../../kittens/1.jpg" alt="cat 1"
/><span>Chance</span></li>
18       <li><img src="../../kittens/2.jpg" alt="cat 2"
/><span>MeowMix</span></li>
19       <li><img src="../../kittens/3.jpg" alt="cat 3"
/><span>Stripes</span></li>
20       <li><img src="../../kittens/4.jpg" alt="cat 4"
/><span>Schrodinger</span></li>
21       <li><img src="../../kittens/5.jpg" alt="cat 5"
/><span>Waffles</span></li>
22       <li><img src="../../kittens/6.jpg" alt="cat 6"
/><span>Tazmina</span></li>
23       <li><img src="../../kittens/7.jpg" alt="cat 7"
/><span>Wiggle</span></li>
24       <li><img src="../../kittens/8.jpg" alt="cat 8"
/><span>Gin</span></li>
25       <li><img src="../../kittens/9.jpg" alt="cat 9"
/><span>Paws</span></li>
26       <li><img src="../../kittens/10.jpg" alt="cat 10"
/><span>Marshmallow</span></li>
27       <li><img src="../../kittens/11.jpg" alt="cat 11"
/><span>Caesar</span></li>
28       <li><img src="../../kittens/12.jpg" alt="cat 12"
/><span>Trogdor</span></li>
29       <li><img src="../../kittens/13.jpg" alt="cat 13"
/><span>Sourpuss</span></li>
30       <li><img src="../../kittens/14.jpg" alt="cat 14"
/><span>Stitch</span></li>
31       <li><img src="../../kittens/15.jpg" alt="cat 15"
/><span>Sunshine</span></li>
32     </ul>
33     <p>Choose the cutest cats. Pick three winners!</p>
34   </section>
35
36   <section id="cutest_section">
37     <h1>Cutest Cats</h1>
38     <ul id="cutest" class="kittenlist">
39     </ul>
40     <p>Click to remove</p>
```

```
41   </section>
42
43 </body>
44 </html>
```

Codebeispiel 12.5 *additional_files/12/solutions/cude_kittens/cutycat.html*

```
1 h1 {
2   font-size: 5em;
3   margin: 0;
4   padding: 0;
5 }
6
7 p {
8   font-size: 2em;
9   font-style: italic;
10 }
11
12 .kittenlist {
13   border: 1px solid black;
14   width: 700px;
15   margin: 0;
16   padding: 0;
17   height: 80vh;
18   overflow: auto;
19 }
20
21 .kittenlist li {
22   list-style: none;
23   font-size: 5em;
24   background-color: #055;
25   padding: 5px;
26   margin-bottom: 20px;
27   height: 150px;
28 }
29
30 .kittenlist li:hover {
31   background-color: #755;
32   outline: 5px solid #700;
33 }
34
35 .kittenlist li img {
36   margin-right: 20px
37 }
38
39 .kittenlist li span {
40   display: inline;
41   top: -0.7em;
42   position: relative;
43   color: #099;
44 }
45
46 #cutest {
47   height: 520px;
48   overflow: hidden;
```

```
49 }
50
51 #cutest_section {
52   position: absolute;
53   top: 0;
54   left: 50vw;
55 }
```

Codebeispiel 12.6 *additional_files/12/solutions/cude_kittens/cutycat.css*

2. Achten Sie darauf, dass sich die gleiche Katze nicht zweimal auswählen lässt.

3. Es darf nicht mehr als 3 niedlichste Katzen geben.

4. Benutzer sollen eine Katze in der Gewinnerbox (*Cutest Cats*) anklicken können, um sie wieder herauszunehmen.

Hinweise

- Versuchen Sie die Übung ohne `.innerHTML` zu lösen!
- Unter der URL *loremflickr.com/150/150/cute,kitten/all* können Sie weitere zufällige Katzenbilder generieren.
- Die Seite *fantasynamegenerators.com/cat-names.php* produziert zufällige Katzennamen.
- Verwenden Sie `.clone(true)` auf den `li`-Elementen, um das Entfernen der Katzen aus *Our Cats* zu verhindern.
- Registrieren Sie den Event-Handler auf den `li`-Elementen. Falls der Benutzer aber nicht direkt das `li` anklickt, sondern Bild oder Text (Name der Katze), so zeigt `event.target` auf das angeklickte Element. Mittels `event.currentTarget` können Sie aber immer noch auf das `li` zugreifen. `currentTarget` enthält immer das Element, auf dem der Handler tatsächlich registriert wurde.

13 Anhang A: Referenz

Diese Referenz fasst die Browser-APIs zusammen, soweit dieser Kurs sie vermittelt. Auf selten benötigte oder zu komplexe Eigenschaften & Methoden haben wir bewusst verzichtet.

Verwenden Sie diese Referenz als Nachschlagewerk für Ihre Übungen. Wir hoffen, dass sie Ihnen auch später bei der täglichen Arbeit eine Hilfe ist. Sobald Sie die Namen der grundlegenden Methoden, Attribute und Interfaces aber halbwegs kennen, macht es vermutlich mehr Sinn, auf eine Online-Referenz wie *developer.mozilla.org* oder *www.webplatform.org* umzusteigen. Falls Sie wissen möchten, ab welchen Browserversionen eine bestimmte Funktionalität zur Verfügung steht, empfehlen wir Ihnen *caniuse.com*.

Mein Tipp

Benutzen Sie die Referenz nicht nur als Nachschlagewerk. Lesen Sie die Referenz einfach mal bei Gelegenheit — wie eine Sonntagszeitung oder den aktuellen Twitter-Feed.

Lesen Sie nicht unbedingt hochkonzentriert und fokussiert, sondern einfach zum Entspannen. Stellen Sie sich dabei vor, was Sie mit den Attributen und Methoden für spannende Dinge programmieren könnten. Und bei Gelegenheit: Lesen Sie die Referenz nochmal.

Auf diese Weise prägen Sie sich die wichtigsten Dinge quasi »im Vorbeigehen« ein, und Sie haben alles schon einmal gesehen. Falls Sie dann später z. B. eine bestimmte Methode benötigen, fällt Ihnen sofort wieder ein, dass es da etwas Passendes gibt.

Referenzen zu lesen, muss wirklich nicht »trocken« sein. Als ich mit 12 mein erstes Programmierbuch las (das Basic-Handbuch für den Commodore C64), hab' ich den Referenzteil geradezu »verschlugen«. Bei jeder neuen Funktion hab' ich davon geträumt, welche großartigen Dinge ich mal damit programmieren würde.

13.1 Elemente

Hinweis: Die im DOM zur Verfügung stehenden Methoden stammen aus verschiedenen Interfaces/APIs. Diese Herkunfts-APIs sind in den folgenden Referenztabellen jeweils mit aufgeführt. Das hat rein dokumentarische Gründe. Sie können bei der entsprechenden API nachlesen, welche Methoden nun genau zur Verfügung stehen.

13.1.1 Element vs. Node

Das DOM unterscheidet zwischen Knoten (***Node***) im Allgemeinen und speziellen Knotentypen, wie Element-Knoten (***Element***), Text-Knoten, Kommentar-Knoten usw. Für die Praxis sind normalerweise nur die Element-Knoten relevant. Daher konzentriert sich unser Kurs auf diese. Falls Sie sich doch für weitere, eher selten benötige Knotentypen interessieren, finden Sie unter *nodeType im MDN*[41] eine Auflistung mit Erläuterungen.

Da ein Element-Knoten (*Element*) eine Spezialisierung eines Knotens (*Node*) darstellt, stehen Ihnen auf dem Element-Knoten neben den Element-Eigenschaften (z. B. `children`) auch alle Node-Eigenschaften (z. B. `parentNode`) zur Verfügung. Im Normalfall können Sie den Unterschied ignorieren.

13.1.2 Der Document-Knoten

Der `document`-Knoten ist ebenfalls ein Spezialfall eines Knotens. Er repräsentiert das komplette HTML-Dokument und stellt einige zusätzliche Methoden wie `createElement` bereit.

Die Methoden `querySelector` und `querySelectorAll` lassen sich auf `document` **global** und auf einzelnen Elementen **lokal** verwenden. Global heißt, der Selektor durchsucht das gesamte Dokument. Lokal bedeutet, dass nur die Nachfahren (d. h. Kinder, Enkel usw.) des entsprechenden Element-Knotens zur Suche herangezogen werden.

41. *https://developer.mozilla.org/en/docs/Web/API/Node/nodeType*

13.1.3 DOM Selection

Methode	Beschreibung	Interface/API
querySelector	Gibt das **erste** Element zurück, das zum angegeben Selektor passt. Durchsucht nur Elemente innerhalb des HTML-Teilbaums, auf dessen Wurzelelement `querySelector` aufgerufen wurde.	Element
querySelectorAll	Gibt **alle** Elemente zurück, die zum angegeben Selektor passen. Durchsucht nur Elemente innerhalb des HTML-Teilbaums, auf dessen Wurzelelement `querySelectorAll` aufgerufen wurde.	Element
document.querySelector *(aka $)*	Gibt das **erste** Element zurück, das zum angegeben Selektor passt. Durchsucht das gesamte HTML-Dokument.	Document
document.querySelectorAll *(aka $$)*	Gibt **alle** Elemente zurück, die zum angegeben Selektor passen. Durchsucht das gesamte HTML-Dokument.	Document

Tabelle 13.1 *Diese Tabelle zeigt die wichtigsten Methoden zur* ***Selektion*** *von DOM-Elementen. Weitere finden Sie im MDN*[42] *bei den Beschreibungen der entsprechenden APIs: Element*[43] *& Document*[44].

42. *https://developer.mozilla.org*
43. *https://developer.mozilla.org/en-US/docs/Web/API/Element*
44. *https://developer.mozilla.org/en-US/docs/Web/API/Document*

Attribute	Beschreibung	Interface/API
document.head	Gibt das `head`-Element zurück.	Document
document.body	Gibt das `body`-Element zurück.	Document

Tabelle 13.2 *Diese Tabelle zeigt die wichtigsten Attribute zur* ***Selektion*** *von DOM-Elementen. Weitere finden Sie im MDN*[45] *bei den Beschreibungen der Document-API.*

CSS Selektoren

Name	Beispiel	Beschreibung
Universal selector	`*`	Selektiert ein beliebiges Element. Diesen Selektor sollten Sie nicht ohne weitere Einschränkungen verwenden, da das Selektieren **aller** Elemente einer Website sehr viel Rechenzeit in Anspruch nehmen kann.
Type selector	`h1`	Selektiert alle Elemente eines bestimmten Typs.
ID selector	`#some-id`	Selektiert genau **ein** Element mit der gegebenen `id`.
Class selector	`.some-class`	Selektiert alle Elemente, denen die angegebene Klasse zugewiesen wurde. Beachten Sie, dass ein Element mehrere Klassen haben kann.
-	`h1, p, .go`	Das Komma ist selbst kein Selektor, sondern erlaubt es, mehrere Selektoren zu kombinieren. Im Beispiel würden alle `h1`-Elemente, alle `p`-Elemente und alle Elemente mit der Klasse `go` selektiert.

Tabelle 13.3 *Basis CSS3-Selektoren*

45. *https://developer.mozilla.org*

Selektor	Beispiel zu `<img src="funny_cat.png" ... />`	Beschreibung
[*attr*]	`[src]`	Selektiert Elemente mit dem angegebenen Attribut. Der Attributwert spielt dabei keine Rolle (***attribute presence***).
[*attr=val*]	`[src='funny_cat.png']`	Selektiert Elemente, die das angegebene Attribut (*src*) mit genau dem angegebenen Wert (*funny_cat.png*) enthalten (***exact attribute match***).
[*attr*=val*]	`[src*=cat]`	Selektiert Elemente, bei denen das angegebene Attribut (*src*) den Wert (*cat*) als Teilstring enthält (***substring attribute match***).
[*attr^=val*]	`[src^=funny]`	Selektiert Elemente, bei denen das angegebene Attribut (*src*) mit dem Wert (*funny*) beginnt.
[*attr$=val*]	`[src$=png]`	Selektiert Elemente, bei denen das angegebene Attribut (*src*) auf den angegebenen Wert (*png*) endet.

Tabelle 13.4 *CSS attribute selectors. All examples select *

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <title>Selector Testing</title>
</head>

<body>
  <ul id="ul1">
    <li id="li11"><span>11</span></li>
    <li id="li12">12</li>
    <li id="li13">13</li>
    <li id="li14">14</li>
  </ul>
  <ul id="ul2">
    <li id="li21"><span>21</span></li>
    <li id="li22"><strong><span>22</span></strong></li>
    <li id="li23">23</li>
```

```
  </ul>
</body>

</html>
```

Codebeispiel 13.1 *additional_files/02/examples/selector_testing.html*

Name	Beispiel	Beschreibung
Descendant combinator (Leerzeichen)	`li span`	Selektiert Elemente (hier `span`), die die Nachfahren eines anderen angegebenen Elements (hier `li`) sind — völlig unabhängig davon, wie viele Ebenen dazwischen liegen. **Ergebnis:** `[<span>11</span>, <span>21</span>, <span>22</span>]`
Adjacent sibling combinator (+)	`li#li11 + li`	Mit diesem Selektor können Sie das Element auswählen, das im HTML-Quelltext **direkt nach** dem angegeben Element (hier `li#li11`) auf der gleichen Ebene (Geschwister-Element) folgt. Falls das im Dokument nachfolgende Element nicht dem hinter dem +-Symbol angegebenen entspricht, wird nichts selektiert. **Ergebnis:** `[<li id="li12">12</li>]`

Tabelle 13.5 *Combinator Selectors — Ergebnisse basieren auf Codebeispiel 13.1*

Name	Beispiel	Beschreibung
General sibling combinator (~)	`li#li12 ~ li`	Mit diesem Selektor können Sie alle Elemente auswählen, die im Dokument nach dem angegebenen Element (hier `li#li12`) auf der gleichen Ebene (Geschwister-Element) folgen. Es wird **nicht nur das direkt folgende** Element gewählt, wie beim *Next sibling combinator*. **Ergebnis:** `[<li id="li13">13</li>, <li id="li14">14</li>]`
Child combinator (>)	`li > span`	Eine Variante der Nachfahren-Selektoren sind die Kind-Selektoren, mit denen Sie Elemente auswählen können, die **direkte** »Kinder« von übergeordneten Eltern-Elementen sind. Im Beispiel werden nur `span`-Elemente selektiert, die sich direkt innerhalb eines `li`-Elements befinden. `<span>22</span>` befindet sich in einem `strong`-Element und wird deswegen nicht selektiert. **Ergebnis:** `[<span>11</span>, <span>21</span>]`

Tabelle 13.5 *Combinator Selectors — Ergebnisse basieren auf Codebeispiel 13.1*

Name	Beispiel	Beschreibung
:first-child	ul li:first-child	Mit dieser Pseudo-Klasse sprechen Sie das erste Kind-Element eines übergeordneten Containers (hier: `ul`) an, falls es sich um ein Element vom angegeben Tag-Typ (hier: `li`) handelt. Sollte das erste Kindelement nicht den richtigen Tag-Typ haben, wird nichts selektiert.
:nth-child(x)	ul li:nth-child(5)	Die Pseudo-Klasse spricht das x-te Kind-Element an — im Beispiel das fünfte Element innerhalb von `ul`, falls es vom Tag-Typ `li` ist.
:last-child	ul li:last-child	Äquivalent zu `:first-child` selektieren Sie hier das letzte Kind-Element, sofern es vom richtigen Tag-Typ ist.

Tabelle 13.6 *Pseudo Classes*

Name	Beispiel	Beschreibung
:only-child	#article p:only-child	Diese Klasse selektiert ein Element nur, wenn es sich um das einzige Kind eines angegebenen Eltern-Elementes handelt.
:empty	div:empty	Mit dieser Pseudo-Klasse sprechen Sie nur Elemente an, die ohne Kind-Elemente sind. Inhalt in Form von Text wird dabei ebenfalls als Kind-Element betrachtet.
:not(selector)	:not(span)	Negation. Wählt Elemente aus, wenn sie nicht dem in Klammern angegebenen Selektor entsprechen. Im Beispiel würde alles außer `span`-Elementen ausgewählt.

Tabelle 13.6 *Pseudo Classes*

13.1.4 DOM Manipulation

Methode	Beschreibung	API
appendChild	Hängt einen übergebenen Knoten (z. B. ein Element) als letztes Kind an den Elternknoten an, auf dem die Methode aufgerufen wurde. Falls der angehängte Knoten bereits Teil des Dokuments war, wird er von seiner ursprünglichen Position entfernt.	Node
insertBefore	Fügt einen übergebenen Knoten (erstes Argument) vor einem anderen Knoten (zweites Argument) ein. Ausgegangen wird dabei von dem Elternknoten, auf dem die Methode aufgerufen wurde.	Node
remove	Entfernt einen Knoten aus seinem HTML-Baum. Der Knoten wird zurückgegeben und kann weiterverwendet werden.	ChildNode

Tabelle 13.7 *Diese Tabelle zeigt die wichtigsten Methoden zur* ***Manipulation*** *von DOM-Elementen. Weitere finden Sie im MDN*[46] *bei den Beschreibungen der entsprechenden APIs: Node & ChildNode.*

46. *https://developer.mozilla.org*

13.1.5 DOM Traversal

Um sich innerhalb des DOM von einem Element zum anderen zu bewegen (***traversal***), sind verschiedene Attribute nötig; dieser Referenzteil zeigt die wichtigsten.

Alle in Tabelle 13.8 bzw. Abb. 13.1 aufgelisteten Attribute sind *read-only*. Sie können zwar anhand der Attribute im DOM navigieren, aber keine Veränderungen an der Baumstruktur durchführen.

Attribut	Beschreibung	API
parentNode	der Elternknoten	Node
children	alle Kindelemente	parentNode
childElementCount	Anzahl der Kindelemente	parentNode
firstElementChild	das erste Kindelelement	parentNode
lastElementChild	das letzte Kindelelement	parentNode
previousElementSibling	das vorhergehende Geschwisterelement	NonDocumentTypeChildNode
nextElementSibling	das nächste Geschwisterelement	NonDocumentTypeChildNode

Tabelle 13.8 *Diese Tabelle zeigt die wichtigsten Attribute zum DOM-**Traversal** (Bewegung innerhalb des DOMs). Weitere finden Sie im MDN[47] bei den Beschreibungen der entsprechenden APIs.*

47. *https://developer.mozilla.org*

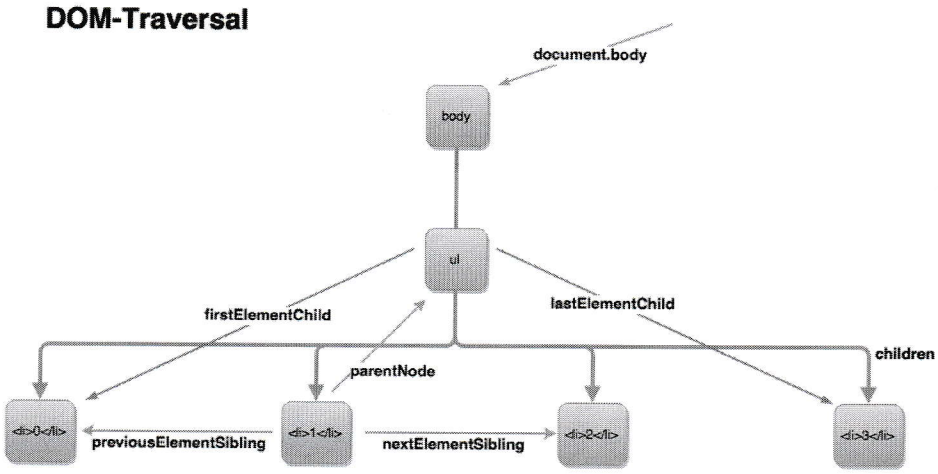


Abb. 13.1 *DOM-Traversal-Attributes*

13.1.6 DOM Creation

Methode	Beschreibung	API
cloneNode	Kopiert einen Knoten und gibt die Kopie zurück. `cloneNode(true)` kopiert auch alle Kindknoten mit.	Node
document.createElement	Erzeugt ein Element vom angegebenen Typ	Document

Tabelle 13.9 *Diese Tabelle zeigt die wichtigsten Methoden zur Erzeugung (**Creation**) von DOM-Elementen. Weitere finden Sie im MDN[48] bei den Beschreibungen der entsprechenden APIs.*

48. *https://developer.mozilla.org*

Attribut	Beschreibung	API
innerHTML	Gibt das HTML im Inneren eines Elementes zurück — der Wert kann auch verändert werden.	Element
textContent	Gibt den Text (ohne HTML-Tags) zurück, der sich im HTML-Element befindet — der Wert kann auch verändert werden.	Node

Tabelle 13.10 *Diese Tabelle zeigt die wichtigsten Attribute zur Erzeugung (**Creation**) von DOM-Elementen. Weitere finden Sie im MDN[49] bei den Beschreibungen der entsprechenden APIs.*

13.2 Attribute

Name des Attributs	Elemente, für die das Attribut oft verwendet wird	Beschreibung
alt	`area, img, input`	Alternativer Text, z. B. falls ein Bild nicht dargestellt werden kann.
checked	`input`	Gibt an, ob ein Radiobutton gewählt oder eine Checkbox angehakt ist.
cols	textarea	Anzahl der Spalten
disabled	`button, input, option, select, textarea`	Gibt an, ob ein Element deaktiviert ist.
href	`a, link`	Die URL einer verlinkten Ressource

Tabelle 13.11 *Ein Auswahl von Attributen, die in der Praxis öfter mit JS angesprochen werden. Eine vollständige Liste aller Attribute finden Sie in der MDN Attribut reference[50].*

49. *https://developer.mozilla.org*
50. *https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes*

Name des Attributs	Elemente, für die das Attribut oft verwendet wird	Beschreibung
max	`progress`, `input`	Der Maximalwert — bei `input`-Elementen ist das nur für `type="number"` und `type="range"` sinnvoll.
maxlength	`input`, `textarea`	Maximal erlaubte Zeichen in einem Element
min	`input`	Der Minimalwert — bei `input`-Elementen ist das nur für `type="number"` und `type="range"` sinnvoll.
multiple	`input`, `select`	Erlaubt bei `true` mehrere Eingaben für `input type="email"` oder `input type="file"`. Bei `select` können mit **strg/cmd** mehrere Optionen gewählt werden.
name	`button`, `form`, `fieldset`, `input`, `select`, `textarea`	Name des Elements. Dieses Attribut wird hauptsächlich für serverseitige Anwendungen zur Identifizierung genutzt.
placeholder	`input`, `textarea`	Voreingestellter Text, der dem Anwender einen Hinweis zu den erwarteten Eingaben gibt.
readonly	`input`, `textarea`	Gibt an, ob ein Element bearbeitet werden kann.
rel	`a`, `link`	Definiert die Beziehung zwischen dem Ziel- und dem Link-Objekt. Kann auch für eigene Beziehungstypen verwendet werden, z. B. interne Links für eine JS-Bildergalerie.
required	`input`, `select`, `textarea`	Gibt an, ob ein Element ein Pflichtfeld darstellt.

Tabelle 13.11 *Ein Auswahl von Attributen, die in der Praxis öfter mit JS angesprochen werden. Eine vollständige Liste aller Attribute finden Sie in der MDN Attribut reference.*

Name des Attributs	Elemente, für die das Attribut oft verwendet wird	Beschreibung
rows	`textarea`	Anzahl der Zeilen
selected	`option`	Gibt an, ob eine `option` selektiert ist.
size	`select`	Anzahl der sichtbaren Optionen bei einer `select`-Box
src	`img`	URL der Bilddatei
start	`ol`	Definiert die Startnummer einer Liste, wenn etwas anderes als 1 gewünscht ist.
step	`input`	Die Schrittweite — bei `input`-Elementen ist das nur für `type="number"` und `type="range"` sinnvoll.
tabindex	auf allen Elementen möglich	Überschreibt die herkömmliche Reihenfolge der angesteuerten (Formular-)Elemente mit Hilfe der Tab-Taste.
title	auf allen Elementen möglich	Kleine Text-Box (Tooltip), die beim »Hover« angezeigt wird.
type	`button`, `input`	Typ des Elements
value	`button`, `option`, `input`, `progress`	Wert des Elements

Tabelle 13.11 *Ein Auswahl von Attributen, die in der Praxis öfter mit JS angesprochen werden. Eine vollständige Liste aller Attribute finden Sie in der MDN Attribut reference.*

13.3 CSS

Methode	Beispiel	Beschreibung
add	$("#bar").classList.add("foo")	Fügt eine Klasse hinzu. Im Beispiel erhält das Element mit der `id` *bar* die Klasse *foo*.
remove	$("#bar").classList.remove("foo")	Entfernt eine Klasse. Enfernt im Beispiel die Klasse *foo* von dem Element mit der `id` *bar*.
item	$("#bar").classList.item(2)	Gibt eine Klasse aus der Classlist anhand des übergebenen Index-Arguments zurück. Gibt im Beispiel den dritten Klassennamen des Elements mit der `id` *bar* zurück.
toggle	$("#bar").classList.toggle("foo")	Entfernt eine Klasse oder fügt sie hinzu. Im Beispiel wird die Klasse *foo* hinzugefügt, falls das Element mit der `id` *bar* noch nicht über diese Klasse verfügt. Falls die Klasse schon vorhanden ist, wird sie entfernt.
contains	$("#bar").classList.contains("foo")	Gibt `true` zurück, falls die angegebene Klasse in der Classlist vorhanden ist.

Tabelle 13.12 *Methoden des classList-Interfaces. Ein ausführliche Referenz finden Sie im MDN*[51]

51. *https://developer.mozilla.org/en/docs/Web/API/Element/classList*

Attribut	Beispiel	Beschreibung
style	$("#bar").style	Enthält die CSS-Stile des Elements als *CSSStyle-Declaration* (änderbar)
classList	$("#bar").classList	Enthält die Liste der CSS-Klassen als *DOMTo-kenList* (änderbar).

Tabelle 13.13 *Attribute des HTMLElement-Interfaces für CSS-Zugriff. Ein ausführliche Referenz finden Sie im MDN*[52]

13.4 Events

Eventname	wird ausgelöst, wenn ...	z. B. verfügbar auf
blur	ein Element den Fokus verliert.	input, textarea
change	ein Anwender eine Änderung am Element bewirkt — z. B. durch Wählen einer Option innerhalb einer `select`-Box oder durch das Bestätigen einer Checkbox. Im Gegensatz zum `input`-Event gilt: Bei einem `input`-Field oder einer `textarea` »feuert« dieses Event erst, nachdem das Feld seinen Fokus wieder verliert (d. h. die Eingabe als abgeschlossen gilt).	input, select, textarea
click	ein Anwender auf ein Element klickt (und wieder loslässt). Das `click`-Event repräsentiert den kompletten Vorgang: Drücken und Loslassen. Wenn Sie nur auf das Runterdrücken reagieren möchten, verwenden Sie `mousedown`, für nur das Loslassen `mouseup`.	button, img, body, p, div

Tabelle 13.14 *Ein Auswahl in der Praxis benötigter Browser-Events. Eine vollständige Liste finden Sie in der MDN Event reference*[53].

52. *https://developer.mozilla.org/en/docs/Web/API/Element/classList*
53. *https://developer.mozilla.org/en-US/docs/Web/Events*

Eventname	wird ausgelöst, wenn ...	z. B. verfügbar auf
dblclick	ein Anwender auf einem Element einen Doppelklick ausführt.	button, img, body
focus	ein Element den Fokus erhält.	input, textarea
hashchange	sich der Fragmentbezeichner der URL ändert (der Fragmentbezeichner ist der Teil, der nach dem #-Symbol folgt, inkl. Symbol).	window
input	der Inhalt eines `input`- oder `textarea`-Elements geändert wird. Im Gegensatz zu `change` feuert dieser Event bei jeder Eingabe sofort. Unter *jsfiddle.net/AtvtZ/* finden Sie einen guten Vergleich von `input` und `change`.	input, textarea
keydown	der Anwender eine Taste drückt.	input, textarea
keypress	der Anwender eine Taste gedrückt hält.	input, textarea
keyup	der Anwender eine Taste loslässt.	input, textarea
mouseenter	der Anwender den Mauszeiger über das Element bewegt — genauer: die Fläche des Elements mit dem Mauszeiger betritt.	button, img, body, p, div
mouseleave	der Anwender den Mauszeiger aus einem Element herausbewegt.	button, img, body, p, div
mousemove	sich der Mauszeiger über dem Element bewegt.	button, img, body, p, div
reset	ein Formular zurückgesetzt wird.	form

Tabelle 13.14 *Ein Auswahl in der Praxis benötigter Browser-Events. Eine vollständige Liste finden Sie in der MDN Event reference.*

Eventname	wird ausgelöst, wenn ...	z. B. verfügbar auf
resize	der Anwender die Größe des Browserfensters verändert.	window
scroll	innerhalb eines Elements (oder des Browserfensters) gescrollt wird.	window, div
select	der Anwender Text markiert.	input, textarea, p
submit	ein Formular abgeschickt wird.	form
transitionend	eine CSS-Transition beendet ist.	img, div, p
visibilitychange	Inhalt eines Elements sichtbar oder unsichtbar wird.	img, div, p
wheel	der Anwender am Rad dreht (wir meinen das wörtlich, nicht im übertragenen Sinn). Normalerweise ist es das Mausrad, aber es kann auch ein Trackball oder ein Touchpad sein.	window, div

Tabelle 13.14 *Ein Auswahl in der Praxis benötigter Browser-Events. Eine vollständige Liste finden Sie in der MDN Event reference.*

Anhang B: Quellen & Literaturhinweise 14

14.1 APA-Style

Zum Aufzeigen von Referenzen, Quellen und weiterführender Literatur verwenden wir den sogenannten ***APA-Style***. Der APA-Style ist — unter anderem — ein System zur Kennzeichnung von Referenzen. Entwickelt wurde das System von der *American Psychological Association* (APA).

Weitere Informationen dazu finden Sie auf *Wikipedia*[54] oder im *Online Writing Lab*[55]. Eine APA-Style-Referenz besteht meist aus Nachname und Jahreszahl. Hier sehen Sie zwei Beispiele.

> *Ein Indiz für die Aussagekraft eines Bezeichners ist auch seine Länge. Variablen, die aus nur einem Zeichen bestehen, sind meistens problematisch (Kellerwessel 2002).*

> *Douglas Crockford (2008) bezeichnet das* `==` *sogar als bösen Zwilling.*

Im folgenden Quellenverzeichnis können Sie die Referenzen dann nachschlagen.

14.2 Quellen

Crockford D. (2001). JavaScript: The World's Most Misunderstood Programming Language. Douglas Crockford's Wrrrld Wide Web (private Website). Betrachtet am 19.03.2015 unter *javascript.crockford.com/javascript.html*

Crockford D. (Mai 2008). JavaScript: The Good Parts: Working with the Shallow Grain of JavaScript. O'Reilly

Ecma International (Juni 2015). ECMAScript 2015 Language Specification. Standard ECMA-262, 6th Edition / June 2015

Ecma International (Juni 2011). ECMAScript Language Specification. Standard ECMA-262, 5.1 Edition / June 2011

54. *http://en.wikipedia.org/wiki/Apa_style*
55. *https://owl.english.purdue.edu/owl/resource/560/01/*

Fowler M., Beck K., Brant J., Opdyke W. et. al. (1999). Refactoring: Improving the Design of Existing Code, Amsterdam: Addison-Wesley Longman

Keith, J. (2005). DOM Scripting: Web Design with JavaScript and the Document Object Model, 1st Edition, Friends of Ed

Martin R. C. (2002). Agile Software Development. Principles, Patterns, and Practices. Prentice Hall

Martin R. C. (2008). Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall

Nicholus R. (2015). Beyond jQuery. Beta Book 2015-09-25. Lean Pub

Opdyke W. F., Johnson R. E. (September 1990). Refactoring: An Aid in Designing Application Frameworks and Evolving Object-Oriented Systems. Proceedings of the Symposium on Object Oriented Programming Emphasizing Practical Applications (SOOPPA). ACM

Osmani A. (2015). Learning JavaScript Design Patterns. O'Reilly

Rauschmayer A. (April 2016). Setting up ES6. Leanpub

Venners B. (März 2003). Orthogonality and the DRY Principle - A Conversation with Andy Hunt and Dave Thomas, Part II. Interview in einem Web-Artikel. Betrachtet am 12.04.2016 unter *www.artima.com/intv/dry.html*

Lösungen der Übungsaufgaben

Übung 1: Almost Famous Quotes

1. ```
   document.querySelector('h1').innerHTML = 'Almost Famous Quotes';
   ```

2. ```
   document.querySelector('blockquote').innerHTML = '"<p>I have always wished for my computer to be as easy to use as my telephone; my wish has come true because I can no longer figure out how to use my telephone.</p><footer>— <cite>Bjarne Stroustrup</cite></footer>';
   ```

Übung 2: 01001000010011110101010 0 — Teil 1

1. document.querySelector('h1')
2. document.querySelector('#buy_form')
3. document.querySelector('#product_img')

Übung 3: 01001000010011110101010 0 — Teil 2

1. document.querySelectorAll('li')
2. document.querySelectorAll('h2')
3. document.querySelectorAll('.special')
4. document.querySelectorAll('li.keyword')
5. document.querySelectorAll('span.special')
6. document.querySelectorAll('.i.b')

Übung 4: Makler und Anwälte...

1. ```
   const headings = Array.from(document.querySelectorAll('h1'));
   headings.forEach(h => h.innerHTML = 'Lorem ipsum');
   ```

2. ```
   const text = Array.from(document.querySelectorAll('p'));
   text.forEach(p => p.innerHTML = 'Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec,
   ```

```
vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a,
venenatis vitae, justo. Nullam dictum felis eu pede mollis
pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper
nisi.');
```

3.
```
1 const menuItems =
Array.from(document.querySelectorAll('#main_menu a'));
2 menuItems.forEach(a => a.innerHTML = 'Lorem');
```

Übung 5: 01001000010011110101010 0 — Teil 3

1. document.querySelector('ul#product_specification li')
2. document.querySelector('h1.article span')
3. document.querySelectorAll('p .keyword')
4. document.querySelectorAll('ul li')

Übung 6: 01001000010011110101010 0 — Teil 4

1. $$('img[src$=jpg]')
2. $$('form input[type=button]')
3. $$('*.model[data-model*=V7]')
4. $$('img:not([class=float_left]')
5. $$('li:nth-child(2)') oder $$('li:nth-type-of-child(2)')
6. $$('h2 + ul')

Übung 7: 01001000010011110101010 0 — Teil 5

1.

```
Array.from($$('p')).forEach(p => p.classList.add('gray'));
```

2.

```
Array.from($$('p:not(.buy_info_text)')).forEach(p =>
p.classList.add('gray'));
```

3.

```
Array.from($$('li:not([class])')).forEach(li
=>li.classList.add('gray'));
```

Übung 8: Auf die Länge kommt es an???

```
1 const contentLength = () => $("#content").innerHTML.length;
2 const articleSizeCssClass = () =>
3       contentLength() <= 3000 && "coffe_break_article"
4   || contentLength() <= 9000 && "normal_length_article"
5   || "lone_weekend_article";
6 $("h1").classList.add(articleSizeCssClass());
```

Übung 9: Please Click Me

```
1 "use strict";
2
3 {
4   const init = () =>
5     $("#click_me").addEventListener("click", handleButtonClick);
6
7   const handleButtonClick = event =>
8     changeButtonTextTo(event.target,
9         event.ctrlKey
10        ? "Cool, you found an easteregg!"
11        : "Hey I like it when you click me!");
12
13  const changeButtonTextTo = (button, text) => button.innerHTML =
text;
14
15  const $ = document.querySelector.bind(document);
16
17  init();
18 }
```

Codebeispiel 5.6 *additional_files/05/solutions/click_me.js*

Übung 10: Newsboard: Das geht noch besser

1.

```
1 "use strict";
2
3 {
4   const init = () => {
5     showMessageByNumber(currentMessageNumber);
6     $("[title=next]").addEventListener("click", nextMessage);
7     $("[title=prev]").addEventListener("click", prevMessage);
8     $("[title=first]").addEventListener("click", firstMessage);
9     $("[title=last]").addEventListener("click", lastMessage);
10  };
11
12  const nextMessage = e => {
13    showMessageByNumber(currentMessageNumber += 1);
14    e.preventDefault();
15  };
16  const prevMessage = e => {
17    showMessageByNumber(currentMessageNumber -= 1);
```

```
18      e.preventDefault();
19    };
20
21    const firstMessage = e => {
22      showMessageByNumber(currentMessageNumber = 1);
23      e.preventDefault();
24    };
25
26    const lastMessage = e => {
27      showMessageByNumber(currentMessageNumber = messages.length);
28      e.preventDefault();
29    };
30
31    const showMessageByNumber = messageNumber =>
32      $(".newsboard_content").innerHTML = messages[messageNumber -
1];
33
34    const $ = document.querySelector.bind(document);
35
36    const messages = [
37      `<h1>Tutoren streiken!!!</h1>
38      <h2>Alle Einsendeaufgaben werden ab sofort mit 0 Punkten
bewertet</h2>
39      <p>Überall dieselbe alte Leier. Das Layout ist fertig, der
Text lässt auf sich warten. Damit das Layout nun nicht nackt im Raume
steht und sich klein und leer vorkommt, springe ich ein: der
Blindtext. Genau zu diesem Zwecke erschaffen, immer im Schatten
meines großen Bruders »Lorem Ipsum«, freue ich mich jedes Mal, wenn
Sie ein paar Zeilen lesen.</p>
40      <p class="newsboard_footer">am 25.09.2015 von K. Einer</p>`,
41
42      `<h1>Wahnsinn!</h1>
43      <h2>Wie ich mit einer dämlichen Idee ein Vermögen machte</h2>
44      <p>Polyfon zwitschernd aßen Mäxchens Vögel Rüben, Joghurt und
Quark. "Fix, Schwyz! " quäkt Jürgen blöd vom Paß. Victor jagt zwölf
Boxkämpfer quer über den großen Sylter Deich. Falsches Üben von
Xylophonmusik quält jeden größeren Zwerg. Heizölrückstoßabdämpfung.
Zwei flinke Boxer jagen die quirlige Eva und ihren Mops durch
Sylt.</p>
45      <p class="newsboard_footer">am 13.08.2015 von Dr. B.
Lödmann</p>`,
46
47      `<h1>Prokrastination?</h1>
48      <h2>Wie oft hörst du dich selbst sagen: „Nein, ich hab's noch
nicht erledigt; ich habe es vor!"</h2>
49      <p>Denn esse est percipi - Sein ist wahrgenommen werden. Und
weil Sie nun schon die Güte haben, mich ein paar weitere Sätze lang
zu begleiten, möchte ich diese Gelegenheit nutzen, Ihnen nicht nur
als Lückenfüller zu dienen, sondern auf etwas hinzuweisen, das es
ebenso verdient wahrgenommen zu werden: Webstandards nämlich.</p>
50      <p>Sehen Sie, Webstandards sind das Regelwerk, auf dem
Webseiten aufbauen. So gibt es Regeln für HTML, CSS, JavaScript oder
auch XML; Worte, die Sie vielleicht schon einmal von Ihrem Entwickler
gehört haben. Diese Standards sorgen dafür, dass alle Beteiligten aus
einer Webseite den größten Nutzen ziehen. Im Gegensatz zu früheren
Webseiten müssen wir zum Beispiel nicht mehr zwei verschiedene
Webseiten für den Internet Explorer und einen anderen Browser
programmieren.</p>
51      <p class="newsboard_footer">am 02.06.2015 von A. Meisenbär</p>`
```

```
52    ];
53
54    let currentMessageNumber = 1;
55
56    init();
57  }
```

Codebeispiel 5.11 *additional_files/05/solutions/newsboard/newsboard_1a.js*

1. (refactored)

```
1  "use strict";
2
3  {
4    const init = () => {
5      showMessageByNumber(currentMessageNumber);
6      $("[title=next]").addEventListener("click", nextMessage);
7      $("[title=prev]").addEventListener("click", prevMessage);
8      $("[title=first]").addEventListener("click", firstMessage);
9      $("[title=last]").addEventListener("click", lastMessage);
10   };
11
12   const nextMessage = e => showMessageForEvent(e,
currentMessageNumber += 1);
13   const prevMessage = e => showMessageForEvent(e,
currentMessageNumber -= 1);
14   const firstMessage = e => showMessageForEvent(e,
currentMessageNumber = 1);
15   const lastMessage = e => showMessageForEvent(e,
currentMessageNumber = messages.length);
16
17   const showMessageForEvent = (e, targetMessageNumber) => {
18     showMessageByNumber(targetMessageNumber);
19     e.preventDefault();
20   };
21
22   const showMessageByNumber = (messageNumber) =>
23     $(".newsboard_content").innerHTML = messages[messageNumber -
1];
24
25   const $ = document.querySelector.bind(document);
26
27   const messages = [
28     `<h1>Tutoren streiken!!!</h1>
29     <h2>Alle Einsendeaufgaben werden ab sofort mit 0 Punkten
bewertet</h2>
30     <p>Überall dieselbe alte Leier. Das Layout ist fertig, der
Text lässt auf sich warten. Damit das Layout nun nicht nackt im Raume
steht und sich klein und leer vorkommt, springe ich ein: der
Blindtext. Genau zu diesem Zwecke erschaffen, immer im Schatten
meines großen Bruders »Lorem Ipsum«, freue ich mich jedes Mal, wenn
Sie ein paar Zeilen lesen.</p>
31     <p class="newsboard_footer">am 25.09.2015 von K. Einer</p>`,
32
33     `<h1>Wahnsinn!</h1>
34     <h2>Wie ich mit einer dämlichen Idee ein Vermögen machte</h2>
35     <p>Polyfon zwitschernd aßen Mäxchens Vögel Rüben, Joghurt und
Quark. "Fix, Schwyz! " quäkt Jürgen blöd vom Paß. Victor jagt zwölf
```

```
Boxkämpfer quer über den großen Sylter Deich. Falsches Üben von
Xylophonmusik quält jeden größeren Zwerg. Heizölrückstoßabdämpfung.
Zwei flinke Boxer jagen die quirlige Eva und ihren Mops durch
Sylt.</p>
36      <p class="newsboard_footer">am 13.08.2015 von Dr. B.
Lödmann</p>`,
37
38      `<h1>Prokrastination?</h1>
39      <h2>Wie oft hörst du dich selbst sagen: „Nein, ich hab's noch
nicht erledigt; ich habe es vor!“</h2>
40      <p>Denn esse est percipi – Sein ist wahrgenommen werden. Und
weil Sie nun schon die Güte haben, mich ein paar weitere Sätze lang
zu begleiten, möchte ich diese Gelegenheit nutzen, Ihnen nicht nur
als Lückenfüller zu dienen, sondern auf etwas hinzuweisen, das es
ebenso verdient wahrgenommen zu werden: Webstandards nämlich.</p>
41      <p>Sehen Sie, Webstandards sind das Regelwerk, auf dem
Webseiten aufbauen. So gibt es Regeln für HTML, CSS, JavaScript oder
auch XML; Worte, die Sie vielleicht schon einmal von Ihrem Entwickler
gehört haben. Diese Standards sorgen dafür, dass alle Beteiligten aus
einer Webseite den größten Nutzen ziehen. Im Gegensatz zu früheren
Webseiten müssen wir zum Beispiel nicht mehr zwei verschiedene
Webseiten für den Internet Explorer und einen anderen Browser
programmieren.</p>
42      <p class="newsboard_footer">am 02.06.2015 von A. Meisenbär</p>`
43    ];
44
45    let currentMessageNumber = 1;
46
47    init();
48  }
```

Codebeispiel 5.12 *additional_files/05/solutions/newsboard/newsboard_1b.js*

2.

```
1  "use strict";
2
3  {
4    const init = () => {
5      showNumberOfAvailableMessages();
6      showMessageByNumber(currentMessageNumber);
7      $("[title=next]").addEventListener("click", nextMessage);
8      $("[title=prev]").addEventListener("click", prevMessage);
9      $("[title=first]").addEventListener("click", firstMessage);
10     $("[title=last]").addEventListener("click", lastMessage);
11   };
12
13   const showNumberOfAvailableMessages = () =>
14     $(".message_number").innerHTML = messages.length;
15
16   const nextMessage = e => showMessageForEvent(e,
currentMessageNumber += 1);
17   const prevMessage = e => showMessageForEvent(e,
currentMessageNumber -= 1);
18   const firstMessage = e => showMessageForEvent(e,
currentMessageNumber = 1);
19   const lastMessage = e => showMessageForEvent(e,
currentMessageNumber = messages.length);
```

```
20
21    const showMessageForEvent = (e, targetMessageNumber) => {
22      showMessageByNumber(targetMessageNumber);
23      e.preventDefault();
24    };
25
26    const showMessageByNumber = (messageNumber) =>
27      $(".newsboard_content").innerHTML = messages[messageNumber -
1];
28
29    const $ = document.querySelector.bind(document);
30
31    const messages = [
32      `<h1>Tutoren streiken!!!</h1>
33      <h2>Alle Einsendeaufgaben werden ab sofort mit 0 Punkten
bewertet</h2>
34      <p>Überall dieselbe alte Leier. Das Layout ist fertig, der
Text lässt auf sich warten. Damit das Layout nun nicht nackt im Raume
steht und sich klein und leer vorkommt, springe ich ein: der
Blindtext. Genau zu diesem Zwecke erschaffen, immer im Schatten
meines großen Bruders »Lorem Ipsum«, freue ich mich jedes Mal, wenn
Sie ein paar Zeilen lesen.</p>
35      <p class="newsboard_footer">am 25.09.2015 von K. Einer</p>`,
36
37      `<h1>Wahnsinn!</h1>
38      <h2>Wie ich mit einer dämlichen Idee ein Vermögen machte</h2>
39      <p>Polyfon zwitschernd aßen Mäxchens Vögel Rüben, Joghurt und
Quark. "Fix, Schwyz! " quäkt Jürgen blöd vom Paß. Victor jagt zwölf
Boxkämpfer quer über den großen Sylter Deich. Falsches Üben von
Xylophonmusik quält jeden größeren Zwerg. Heizölrückstoßabdämpfung.
Zwei flinke Boxer jagen die quirlige Eva und ihren Mops durch
Sylt.</p>
40      <p class="newsboard_footer">am 13.08.2015 von Dr. B.
Lödmann</p>`,
41
42      `<h1>Prokrastination?</h1>
43      <h2>Wie oft hörst du dich selbst sagen: „Nein, ich hab's noch
nicht erledigt; ich habe es vor!“</h2>
44      <p>Denn esse est percipi - Sein ist wahrgenommen werden. Und
weil Sie nun schon die Güte haben, mich ein paar weitere Sätze lang
zu begleiten, möchte ich diese Gelegenheit nutzen, Ihnen nicht nur
als Lückenfüller zu dienen, sondern auf etwas hinzuweisen, das es
ebenso verdient wahrgenommen zu werden: Webstandards nämlich.</p>
45      <p>Sehen Sie, Webstandards sind das Regelwerk, auf dem
Webseiten aufbauen. So gibt es Regeln für HTML, CSS, JavaScript oder
auch XML; Worte, die Sie vielleicht schon einmal von Ihrem Entwickler
gehört haben. Diese Standards sorgen dafür, dass alle Beteiligten aus
einer Webseite den größten Nutzen ziehen. Im Gegensatz zu früheren
Webseiten müssen wir zum Beispiel nicht mehr zwei verschiedene
Webseiten für den Internet Explorer und einen anderen Browser
programmieren.</p>
46      <p class="newsboard_footer">am 02.06.2015 von A. Meisenbär</p>`
47    ];
48
49    let currentMessageNumber = 1;
50
51    init();
52  }
```

Codebeispiel 5.13 *additional_files/05/solutions/newsboard/newsboard_2.js*

3.

```
1  "use strict";
2
3  {
4    const init = () => {
5      showNumberOfAvailableMessages();
6      showMessageByNumber(currentMessageNumber);
7      $("body").addEventListener("keyup", handleKeyPress);
8      $("[title=next]").addEventListener("click", nextMessage);
9      $("[title=prev]").addEventListener("click", prevMessage);
10     $("[title=first]").addEventListener("click", firstMessage);
11     $("[title=last]").addEventListener("click", lastMessage);
12   };
13
14   const handleKeyPress = e => {
15     if (e.key === "ArrowRight") nextMessage(e);
16     if (e.key === "ArrowLeft") prevMessage(e);
17   };
18
19   const showNumberOfAvailableMessages = () =>
20     $(".message_number").innerHTML = messages.length;
21
22   const nextMessage = e => showMessageForEvent(e,
currentMessageNumber += 1);
23   const prevMessage = e => showMessageForEvent(e,
currentMessageNumber -= 1);
24   const firstMessage = e => showMessageForEvent(e,
currentMessageNumber = 1);
25   const lastMessage = e => showMessageForEvent(e,
currentMessageNumber = messages.length);
26
27   const showMessageForEvent = (e, targetMessageNumber) => {
28     showMessageByNumber(targetMessageNumber);
29     e.preventDefault();
30   };
31
32   const showMessageByNumber = messageNumber =>
33     $(".newsboard_content").innerHTML = messages[messageNumber -
1];
34
35   const $ = document.querySelector.bind(document);
36
37   const messages = [
38     `<h1>Tutoren streiken!!!</h1>
39     <h2>Alle Einsendeaufgaben werden ab sofort mit 0 Punkten
bewertet</h2>
40     <p>Überall dieselbe alte Leier. Das Layout ist fertig, der
Text lässt auf sich warten. Damit das Layout nun nicht nackt im Raume
steht und sich klein und leer vorkommt, springe ich ein: der
Blindtext. Genau zu diesem Zwecke erschaffen, immer im Schatten
meines großen Bruders »Lorem Ipsum«, freue ich mich jedes Mal, wenn
Sie ein paar Zeilen lesen.</p>
41     <p class="newsboard_footer">am 25.09.2015 von K. Einer</p>`,
42
43     `<h1>Wahnsinn!</h1>
```

```
44      <h2>Wie ich mit einer dämlichen Idee ein Vermögen machte</h2>
45      <p>Polyfon zwitschernd aßen Mäxchens Vögel Rüben, Joghurt und
Quark. "Fix, Schwyz! " quäkt Jürgen blöd vom Paß. Victor jagt zwölf
Boxkämpfer quer über den großen Sylter Deich. Falsches Üben von
Xylophonmusik quält jeden größeren Zwerg. Heizölrückstoßabdämpfung.
Zwei flinke Boxer jagen die quirlige Eva und ihren Mops durch
Sylt.</p>
46      <p class="newsboard_footer">am 13.08.2015 von Dr. B.
Lödmann</p>`,
47
48      `<h1>Prokrastination?</h1>
49      <h2>Wie oft hörst du dich selbst sagen: „Nein, ich hab's noch
nicht erledigt; ich habe es vor!“</h2>
50      <p>Denn esse est percipi – Sein ist wahrgenommen werden. Und
weil Sie nun schon die Güte haben, mich ein paar weitere Sätze lang
zu begleiten, möchte ich diese Gelegenheit nutzen, Ihnen nicht nur
als Lückenfüller zu dienen, sondern auf etwas hinzuweisen, das es
ebenso verdient wahrgenommen zu werden: Webstandards nämlich.</p>
51      <p>Sehen Sie, Webstandards sind das Regelwerk, auf dem
Webseiten aufbauen. So gibt es Regeln für HTML, CSS, JavaScript oder
auch XML; Worte, die Sie vielleicht schon einmal von Ihrem Entwickler
gehört haben. Diese Standards sorgen dafür, dass alle Beteiligten aus
einer Webseite den größten Nutzen ziehen. Im Gegensatz zu früheren
Webseiten müssen wir zum Beispiel nicht mehr zwei verschiedene
Webseiten für den Internet Explorer und einen anderen Browser
programmieren.</p>
52      <p class="newsboard_footer">am 02.06.2015 von A. Meisenbär</p>`
53    ];
54
55    let currentMessageNumber = 1;
56
57    init();
58  }
```

Codebeispiel 5.14 *additional_files/05/solutions/newsboard/newsboard_3.js*

4.

```
1  "use strict";
2
3  {
4    const init = () => {
5      showNumberOfAvailableMessages();
6      showMessageByNumber(currentMessageNumber);
7      $("body").addEventListener("keyup", handleKeyPress);
8      $("[title=next]").addEventListener("click", nextMessage);
9      $("[title=prev]").addEventListener("click", prevMessage);
10     $("[title=first]").addEventListener("click", firstMessage);
11     $("[title=last]").addEventListener("click", lastMessage);
12   };
13
14   const handleKeyPress = e => {
15     if (e.ctrlKey) return handleKeyCodeWithCtrl(e);
16     handleKeyCodeWithoutSpecialKeys(e);
17   };
18
19   const handleKeyCodeWithoutSpecialKeys = e => {
20     if (e.key === "ArrowRight") nextMessage(e);
```

```
21      if (e.key === "ArrowLeft") prevMessage(e);
22    };
23
24    const handleKeyCodeWithCtrl = e => {
25      if (e.key === "ArrowRight") firstMessage(e);
26      if (e.key === "ArrowLeft") lastMessage(e);
27    };
28
29    const showNumberOfAvailableMessages = () =>
30      $(".message_number").innerHTML = messages.length;
31
32    const nextMessage = e => showMessageForEvent(e,
currentMessageNumber += 1);
33    const prevMessage = e => showMessageForEvent(e,
currentMessageNumber -= 1);
34    const firstMessage = e => showMessageForEvent(e,
currentMessageNumber = 1);
35    const lastMessage = e => showMessageForEvent(e,
currentMessageNumber = messages.length);
36
37    const showMessageForEvent = (e, targetMessageNumber) => {
38      showMessageByNumber(targetMessageNumber);
39      e.preventDefault();
40    };
41
42    const showMessageByNumber = (messageNumber) =>
43      $(".newsboard_content").innerHTML = messages[messageNumber -
1];
44
45    const $ = document.querySelector.bind(document);
46
47    const messages = [
48      `<h1>Tutoren streiken!!!</h1>
49      <h2>Alle Einsendeaufgaben werden ab sofort mit 0 Punkten
bewertet</h2>
50      <p>Überall dieselbe alte Leier. Das Layout ist fertig, der
Text lässt auf sich warten. Damit das Layout nun nicht nackt im Raume
steht und sich klein und leer vorkommt, springe ich ein: der
Blindtext. Genau zu diesem Zwecke erschaffen, immer im Schatten
meines großen Bruders »Lorem Ipsum«, freue ich mich jedes Mal, wenn
Sie ein paar Zeilen lesen.</p>
51      <p class="newsboard_footer">am 25.09.2015 von K. Einer</p>`,
52
53      `<h1>Wahnsinn!</h1>
54      <h2>Wie ich mit einer dämlichen Idee ein Vermögen machte</h2>
55      <p>Polyfon zwitschernd aßen Mäxchens Vögel Rüben, Joghurt und
Quark. "Fix, Schwyz! " quäkt Jürgen blöd vom Paß. Victor jagt zwölf
Boxkämpfer quer über den großen Sylter Deich. Falsches Üben von
Xylophonmusik quält jeden größeren Zwerg. Heizölrückstoßabdämpfung.
Zwei flinke Boxer jagen die quirlige Eva und ihren Mops durch
Sylt.</p>
56      <p class="newsboard_footer">am 13.08.2015 von Dr. B.
Lödmann</p>`,
57
58      `<h1>Prokrastination?</h1>
59      <h2>Wie oft hörst du dich selbst sagen: „Nein, ich hab's noch
nicht erledigt; ich habe es vor!“</h2>
60      <p>Denn esse est percipi - Sein ist wahrgenommen werden. Und
weil Sie nun schon die Güte haben, mich ein paar weitere Sätze lang
```

```
zu begleiten, möchte ich diese Gelegenheit nutzen, Ihnen nicht nur
als Lückenfüller zu dienen, sondern auf etwas hinzuweisen, das es
ebenso verdient wahrgenommen zu werden: Webstandards nämlich.</p>
61      <p>Sehen Sie, Webstandards sind das Regelwerk, auf dem
Webseiten aufbauen. So gibt es Regeln für HTML, CSS, JavaScript oder
auch XML; Worte, die Sie vielleicht schon einmal von Ihrem Entwickler
gehört haben. Diese Standards sorgen dafür, dass alle Beteiligten aus
einer Webseite den größten Nutzen ziehen. Im Gegensatz zu früheren
Webseiten müssen wir zum Beispiel nicht mehr zwei verschiedene
Webseiten für den Internet Explorer und einen anderen Browser
programmieren.</p>
62      <p class="newsboard_footer">am 02.06.2015 von A. Meisenbär</p>`
63    ];
64
65    let currentMessageNumber = 1;
66
67    init();
68  }
```

Codebeispiel 5.15 *additional_files/05/solutions/newsboard/newsboard_4.js*

Übung 12: Jetzt hat es sich »ausgebuyt«!

```
1  "use strict";
2
3  {
4    const init = () =>
5      $("#buy").addEventListener("click", disableBuyButton);
6
7    const disableBuyButton = e => e.target.disabled = true;
8
9    init();
10 }
```

Codebeispiel 7.5 *additional_files/07/solutions/buy_button/buy_button.js*

Übung 13: Newsboard: Jetzt ist aber Schluss!

```
1  "use strict";
2
3  {
4    const init = () => {
5      initProgressbar();
6      firstMessage();
7      $("[title=next]").addEventListener("click", nextMessage);
8      $("[title=prev]").addEventListener("click", prevMessage);
9      $("[title=first]").addEventListener("click", firstMessage);
10     $("[title=last]").addEventListener("click", lastMessage);
11   };
12
13   const initProgressbar = () => {
14     progressbar().max = messages.length;
15     progressbar().value = 1;
16   };
17
18   const nextMessage = () => {
```

```
19      incCurrentMessageNumber();
20      update();
21    };
22    const prevMessage = () => {
23      decCurrentMessageNumber();
24      update();
25    };
26    const firstMessage = () => {
27      setCurrentMessageNumberToFirstMessage();
28      update();
29    };
30    const lastMessage = () => {
31      setCurrentMessageNumberToLastMessage();
32      update();
33    };
34
35    const incCurrentMessageNumber = () => progressbar().value += 1;
36    const decCurrentMessageNumber = () => progressbar().value -= 1;
37    const setCurrentMessageNumberToFirstMessage = () =>
progressbar().value = 1;
38    const setCurrentMessageNumberToLastMessage = () =>
progressbar().value = progressbar().max;
39
40    const update = () => {
41      updateMessage();
42      updateButtonsState();
43    };
44
45    const updateMessage = () =>
showMessageByNumber(currentMessageNumber());
46
47    const updateButtonsState = () => {
48      buttonNext().disabled = lastMessageReached();
49      buttonPrev().disabled = firstMessageReached();
50      buttonLast().disabled = lastMessageReached();
51      buttonFirst().disabled = firstMessageReached();
52    };
53
54    const lastMessageReached = () => currentMessageNumber() ===
messages.length;
55    const firstMessageReached = () => currentMessageNumber() === 1;
56
57    const showMessageByNumber = messageNumber =>
58      $(".newsboard_content").innerHTML = messages[messageNumber -
1];
59
60    const buttonNext = () => $("[title=next]");
61    const buttonPrev = () => $("[title=prev]");
62    const buttonFirst = () => $("[title=first]");
63    const buttonLast = () => $("[title=last]");
64
65    const currentMessageNumber = () => progressbar().value;
66    const progressbar = () => $("#messages_progress");
67
68    init();
69  }
```

Codebeispiel 7.6 *additional_files/07/solutions/newsboard/newsboard.js*

Übung 14: Die Rückkehr der Lauflichter: Teil 1 - Chasing Lights

```
1  "use strict";
2
3  {
4    const init = () => lights().on("mouseenter", e => {
5      turnAllOff();
6      turnOnNext(e.target);
7    });
8
9    const turnOnNext = light => turnOn(lights()[nextLightNr(light)]);
10   const nextLightNr = light => isLast(light) ? 0 : lightNr(light)
+ 1;
11   const isLast = light => lightNr(light) === lights().length - 1;
12   const lightNr = light => lights().indexOf(light);
13   const turnOn = light => light.src = "light_on.png";
14   const turnOff = light => light.src = "light_off.png";
15   const turnAllOff = () => lights().forEach(turnOff);
16   const lights = () => $$("img[alt=lightbulb]");
17
18   init();
19 }
```

Codebeispiel 7.8 *additional_files/07/solutions/running_light/running_light.js*

Übung 15: Die Rückkehr der Lauflichter: Teil 2 - Triggered Classic

```
1  "use strict";
2
3  {
4    const IMAGE_FILE_ON = "light_on.png";
5    const IMAGE_FILE_OFF = "light_off.png";
6
7    const init = () => document.on("click", () => {
8      const currentLightNr = findFirstTurnedOnLightNr();
9      turnOffByNr(currentLightNr);
10     turnOnByNr(nextLightNr(currentLightNr));
11   });
12
13   const findFirstTurnedOnLightNr = () => lights().findIndex(isOn);
14   const isOn = light => light.src.endsWith(IMAGE_FILE_ON);
15   const nextLightNr = nr => isLast(nr) ? 0 : nr + 1;
16   const isLast = nr => nr === lights().length - 1;
17   const turnOnByNr = nr => turnOn(lights()[nr]);
18   const turnOffByNr = nr => turnOff(lights()[nr]);
19   const turnOn = light => light.src = IMAGE_FILE_ON;
20   const turnOff = light => light.src = IMAGE_FILE_OFF;
21   const lights = () => $$("img[alt=lightbulb]");
22
23   init();
24 }
```

Codebeispiel 7.9 *additional_files/07/solutions/running_light/running_light2.js*

Übung 16: 01001000010011110101010 0 — Teil 6

```
$$('p').forEach(p => p.style.color = "blue");
```

Übung 17: TABula Rasa

```
1  "use strict";
2
3  {
4    const init = () => {
5      registerTabEvents();
6      activateTabByIndex(0);
7    };
8
9    const registerTabEvents = () =>
10     navItems().forEach((elem, i) =>
11       elem.on("click", () => activateTabByIndex(i)));
12
13   const activateTabByIndex = index => {
14     removeHighlightFromActiveNavItem();
15     hightlightNavItemByIndex(index);
16     hideAllArticles();
17     showArticleByIndex(index);
18   };
19
20   const removeHighlightFromActiveNavItem = () =>
21     activeNavItems().forEach(elem =>
22       elem.classList.remove("active"));
23
24   const hightlightNavItemByIndex = index =>
25     navItems()[index].classList.add("active");
26
27   const hideAllArticles = () => articles().forEach(hide);
28
29   const showArticleByIndex = index => show(articles()[index]);
30
31   const articles = () => $$(".tabs article");
32   const navItems = () => $$(".tabs > nav li");
33   const activeNavItems = () => $$(".tabs > nav li.active");
34
35   const show = elem => elem.style.display = "block";
36   const hide = elem => elem.style.display = "none";
37
38   init();
39 }
```

Codebeispiel 8.12 *additional_files/08/solutions/tabbed_navigation/tabs.js*

Übung 18: Farbe Wechsel Dich

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="UTF-8" />
6    <title>Document</title>
7    <script src="../../../lib/dom_helper.js" defer="defer"></script>
8    <script src="color_change.js" defer="defer"></script>
9  </head>
10
11 <body>
12   <button>Red</button>
13   <button>Green</button>
14   <button>Blue</button>
15 </body>
16
17 </html>
```

Codebeispiel 8.13 *additional_files/08/solutions/color_change1/color_change.html*

```
1 "use strict";
2
3 $$("button").on("click", e => $("body").style.backgroundColor =
e.target.innerHTML);
```

Codebeispiel 8.14 *additional_files/08/solutions/color_change1/color_change.js*

Übung 19: Farbe Wechsel Dich — Teil 2

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="UTF-8" />
6    <title>Document</title>
7    <script src="../../../lib/dom_helper.js" defer="defer"></script>
8    <script src="color_change.js" defer="defer"></script>
9  </head>
10
11 <body>
12   Red <input type="range" max="255" id="red"/>
13   Green <input type="range" max="255" id="green" />
14   Blue <input type="range" max="255" id="blue" />
15 </body>
16
17 </html>
```

Codebeispiel 8.15 *additional_files/08/solutions/color_change2/color_change.html*

```
1 "use strict";
2
3 const r = () => $("#red").value;
4 const g = () => $("#green").value;
```

```
5 const b = () => $("#blue").value;
6
7 $$("input[type='range']").on("input", () =>
8   $("body").style.backgroundColor = `rgb(${r()}, ${g()}, ${b()})`);
```

Codebeispiel 8.16 *additional_files/08/solutions/color_change2/color_change.js*

Übung 20: awesome tours

```
1 "use strict";
2
3 const showInfo = e => $("#info").innerHTML = `
4   <h3>
5       <img src="${country(e)}.png" alt="${flagName(e)}"
title="${flagName(e)}" />
6       ${name(e)}
7   </h3>
8   <p>${desc(e)}</p>`;
9
10 const name = e => e.target.alt;
11 const desc = e => e.target.dataset.description;
12 const country = e => e.target.dataset.countryCode;
13 const flagName = e => e.target.dataset.flagName;
14
15 $$("img[data-description]").on("mouseenter", showInfo);
```

Codebeispiel 9.1 *additional_files/09/solutions/awesome_tours/awesome_tours.js*

Übung 21: 01001000010011110101010 0 — Teil 7

1.
```
1 const li = document.createElement("li");
2 li.textContent = "Capacity: 11 oz.";
3 $("#product_specification").appendChild(li);
```

2.
```
1 const option = document.createElement("option");
2 option.textContent = "5 items";
3 $("#buy_form select").appendChild(option);
```

Übung 22: 01001000010011110101010 0 — Teil 8

1.
```
$("h2").remove()
```

2.
```
$("h1 .keyword").remove()
```

Übung 23: 01001000010011110101010 0 — Teil 9

```
1 const li = document.createElement("li");
2 li.textContent = "Capacity: 11 oz.";
3 $("#product_specification").insertBefore(li,
$$("#product_specification li")[4])
```

Übung 24: Cutycat

```
1  "use strict";
2
3  const NUMBER_OF_WINNERS = 3;
4
5  const mayAddCutie = cutie =>
6     $$("#cutest li").length < NUMBER_OF_WINNERS
7     && !isInCutest(cutie);
8
9  const isInCutest = cutie =>
10   $$("#cutest li span")
11     .filter(span => span.textContent === cutie.textContent)
12     .length > 0;
13
14 $$("#candidates li").on("click", e => {
15   const cutie = e.currentTarget;
16   if (!mayAddCutie(cutie)) return;
17
18   $("#cutest").appendChild(
19     cutie.cloneNode(true).on("click", e =>
e.currentTarget.remove())
20   );
21 });
```

Codebeispiel 12.7 *additional_files/12/solutions/cude_kittens/cutycat.js*

Index

A

add 44 48 125 134 151 185
Ajax 167
alt 72 86 87 105 182
API 11 12 42 173 174 175 179 180 181

appendChild 133 136 137 143 144 145

async 53 91

B

Bibliothek 13 14 40 96
Block-Scope 55
blur 84 186

C

change 84 85 186 187
checked 105 182
childElementCount 166 180
children 166 173 180
cloneNode 143 144 145 181
cols 105 182
contains 44 47 49 185
Content 50 100 102 103 104

D

Data 41 129 130 192
dblclick 84 187
defer 53 54 91
disabled 101 102 103 105 161 182
Document Object Model 11 25 190
DOM-Scripting 12
DOM-Traversal 149 151 166 167 181
DOMTokenList 43 44 45 186

E

each 13
ECMA 189
EcmaScript 13 15 189
Eltern-Element 32 155
Eventhandler 64 65 68 71 72 87
Events 63 64 65 72 84 113 123 162 186

F

filter 47 60
firstElementChild 166 180
focus 84 187
Framework 9 12 14 40
Function-Scope 55

G

getComputedStyle 113 116 117 118
Guard Clauses 67
Guard 67

H

hashchange 84 187
high 13 15 60
href 78 105 182

I

IDL 100 102 103 104
IIFE 55
init 69 70 92 101 118 141
Initialisierung 69 78
item 49 185

J

jQuery 13 14 81 190

K

key 71 87 129
KeyboardEvent 71 87
keydown 85 187
keypress 85 187
keyup 64 65 71 72 85 187
keyword 34 38 120 123 129 191 192

L

lastElementChild 166 180

M

map 13 15 60 129
max 98 99 105 183
maxlength 105 183
min 105 183
mouseenter 85 121 123 132 187
mouseleave 85 122 187
mousemove 85 86 123 187
multiple 105 183

N

Newspaper-Metapher 59 60 70
nextElementSibling 156 164 166 180
NodeList 32 43 47 60 61 82

O

on 81 82 83 90 123 162 190

P

Perfomance 118
placeholder 106 183
Polyfill 151
preventDefault 74 79
previousElementSibling 155 156 161
166 180

R

Read-Only 118 166 180
readonly 106 183
rel 106 183
removeChild 151
required 106 183
reset 85 187
resize 85 188

rows 106 184

S

scroll 85 188
select
selected 106 184
selector 27 28 39 175 179
Selektion 42 174 175
siblings 154 155
size 106 111 184
Source-Order 53
start 106 184
step 106 184
strict 54
Style-Objekt 110 114 115 116 117 119

submit 85 188

T

tabindex 107 184
target 72 87 148 155 171
timestamp 72 87
title 78 107 119 184
toggle 44 46 49 185
transitionend 86 188
Traversal 149 151 166 167 180 181
type 27 71 105 106 107 128 175 183
192

V

value 65 98 99 100 101 103 107 184
VanillaJS
visibilitychange 86 188

W

wheel 86 188

JavaScript. HTML mühelos manipuliert.

Die Manipulation von HTML und CSS ist und bleibt die wichtigste Aufgabe von JavaScript im Browser.

Dieses Buch zeigt Ihnen, wie Sie Webseiten für aktuelle Browser aufpeppen und interaktiv gestalten. Sie lernen kleinere Webanwendungen zu bauen, die sich flüssig anfühlen und gut bedienbar sind — statt Anwendungen, die nach jedem Klick die Seite neu aufbauen müssen. Sie lernen, dynamische Tooltips zu erzeugen, Interfaces zum Verwalten von Produkten zu programmieren, eigene Bildgalerien zu entwickeln u.v.m. Lassen Sie Ihre Ideen Realität werden. Wo fertige Webbaukästen Sie nur einschränken, ist JavaScript ein flexibles Werkzeug Ihrer Kreativität. Seit vielen Jahren setzen die Trainer des Europäischen Webmasterverbandes *Webmasters Europe e.V.* die jeweils aktuelle Auflage dieses Buches in der Ausbildung angehender Webentwickler und Webdesigner ein.

Über dieses Lernbuch

Webmasters-Press-Bücher sind *»Lernbücher«*. Sie sind aus der Unterrichtspraxis heraus entwickelt und vermitteln zielstrebig und in klaren Worten Praxiswissen für Ihre eigenen Projekte. Fragen zur Lernerfolgskontrolle und durchdachte, im Schwierigkeitsgrad ansteigende Übungen vertiefen und festigen das neu erworbene Wissen. Zusatzmaterial und Lösungen können Sie auf **webmasters-press.de** herunterladen.

Über die Autoren

Marco Emrich ist Diplom-Informatiker (FH), leidenschaftlicher Trainer und Verfechter der Software-Craftsmanship-Bewegung. Er verfügt über langjährige Erfahrung als Software-Architekt und -Entwickler in ganz unterschiedlichen Branchen. An der Webmasters Akademie in Nürnberg leitet er den Fachbereich Web Engineering. Außerdem hält er regelmäßig Vorträge, leitet Workshops auf bekannten Softwarekonferenzen und schreibt Artikel für Fachzeitschriften. Wenn er in seiner Freizeit nicht gerade Softwerkskammer-Treffen organisiert, erklärt er wahrscheinlich gerade seinem Sohn, wie man Roboterschildkröten programmiert.

Christin Marit ist Diplom-Sozialpädagogin, zertifizierte Webentwicklerin, Fotografin, Bloggerin, Weltenbummlerin und professionelle Lebenskünstlerin. Sie arbeitet für die Webmasters Akademie in Nürnberg als Kursentwicklerin, Autorin und E-Tutorin und findet, dass JavaScript lernen genauso einfach, schön und spannend sein sollte wie das Leben selbst.

ISBN 9781535290722

90000 >

9 781535 290722